STIC Search Report
EIC 2100

USPTO

STIC Database Tracking Number: 170378

TO: Mark Francis
Location: RND 5C11
Art Unit: 2193
Wednesday, November 02, 2005

Case Serial Number: 09/912173

From: Emory Damron
Location: EIC 2100
RND 4B19
Phone: 571-272-3520

Emory.Damron@uspto.gov

Search Notes

Dear Mark,

Please find below your fast and focused search.
References of potential pertinence have been tagged, but please review all the packets in case you like something I didn't.
Of those references which have been tagged, please note any manual highlighting which I've done within the document.
In addition to searching on Dialog, I also searched EPO/JPO/Derwent.
I sent you an email laying out some more info w/ respect to this search.
There may be a few decent references contained herein, but I'll let you determine how useful they may be to you.
Please contact me if I can refocus or expand any aspect of this case, and please take a moment to provide any feedback (on the form provided) so EIC 2100 may better serve your needs. Good Luck!

Sincerely,
Emory Damron
Technical Information Specialist
EIC 2100, US Patent & Trademark Office
Phone: (571) 272-3520
Emory.damron@uspto.gov

sira
Search and Information Resources Administration

USPTO Scientific & Technical Information Center

STIC EIC 2100 Search Request Form

170378

13

Today's Date: 11/2/05

What date would you like to use to limit the search?
Priority Date: 23 July 2001 Other:

Name Mark Francis

AU 2193 Examiner # 80759

Room # 5C11 Phone 27956

Serial # 09912173

Format for Search Results (Circle One):
PAPER DISK EMAIL

Where have you searched so far?
USP DWPI EPO JPO ACM IBM TDB
IEEE INSPEC SPI Other

Is this a "Fast & Focused" Search Request? (Circle One) YES NO
A "Fast & Focused" Search is completed in 2-3 hours (maximum). The search must be on a very specific topic and meet certain criteria. The criteria are posted in EIC2100 and on the EIC2100 NPL Web Page at http://ptoweb/patents/stic/stic-tc2100.htm.

What is the topic, novelty, motivation, utility, or other specific details defining the desired focus of this search? Please include the concepts, synonyms, keywords, acronyms, definitions, strategies, and anything else that helps to describe the topic. Please attach a copy of the abstract, background, brief summary, pertinent claims and any citations of relevant art you have found.

PGPUB 2003/0018641

Claims 17 & 30 (See Attached)

Obtaining performance data of same software run on a first system and a second system.

prioritizing performance data

Obtaining insight on programs and processes, or modules

Obtaining an advice associated with the insight indicating what should be done.

STIC Searcher EMORY DAMRON Phone 2-3520

Date picked up 11/2/05 Date Completed 11/2/05

sira Search and Information Resources Administration

XCOPY

Ref #	Hits	Search Query	DBs	Default Operator	Plurals	Time Stamp
L1	271274	("same" similar identical alike equal equivalent clone) near3 (application software program routine subroutine app apps subprogram macro utility)	US-PGPUB; USPAT; USOCR; EPO; JPO; DERWENT	OR	ON	2005/11/02 12:52
L2	266623	(two different second 2nd another dual twin double pair) near3 (computer server workstation processor processing adj (system device) cpu centralprocessor dataprocessor microprocessor microcomputer)	US-PGPUB; USPAT; USOCR; EPO; JPO; DERWENT	OR	ON	2005/11/02 12:54
L3	2207	1 with 2	US-PGPUB; USPAT; USOCR; EPO; JPO; DERWENT	OR	ON	2005/11/02 12:54
L4	5165	1.clm.	US-PGPUB; USPAT; USOCR; EPO; JPO; DERWENT	OR	ON	2005/11/02 12:54
L6	69461	2.clm.	US-PGPUB; USPAT; USOCR; EPO; JPO; DERWENT	OR	ON	2005/11/02 12:55
L7	202	4 with 6	US-PGPUB; USPAT; USOCR; EPO; JPO; DERWENT	OR	ON	2005/11/02 12:55
L8	132	7 and @ad<"20010724"	US-PGPUB; USPAT; USOCR; EPO; JPO; DERWENT	OR	ON	[illegible]7

PAT LIT

FULL TEXT

vier.com

US005551047A

United States Patent [19]

Mori et al.

[11] **Patent Number:** **5,551,047**

[45] **Date of Patent:** **Aug. 27, 1996**

[54] **METHOD FOR DISTRIBUTED REDUNDANT EXECUTION OF PROGRAM MODULES**

[75] Inventors: **Kinji Mori**, Tokyo; **Masayuki Orimo**, Kawasaki; **Hirokazu Kasashima**, Hitachi, all of Japan; **K. H. Kim**, Irvine, Calif.

[73] Assignee: **The Regents of the Univeristy of California**, Oakland, Calif.

[21] Appl. No.: **10,649**

[22] Filed: **Jan. 28, 1993**

[51] **Int. Cl.**^{6} **G06F 15/16**

[52] **U.S. Cl.** **395/800**; 395/182.09; 364/230.6; 364/260; 364/268.3; 364/DIG. 1

[58] **Field of Search** 395/800, 650, 395/575, 500, 200, 700, 275, 200.03, 200.15, 182.09, 182.11, 183.14, 185.01; 371/9.1, 11.3, 12, 14, 19, 48, 67.1, 68.1, 68.3

[56] **References Cited**

U.S. PATENT DOCUMENTS

4,698,785	10/1987	Desmond et al.	371/19
4,807,228	2/1989	Dahbura et al.	371/9
4,872,106	10/1989	Slater	395/575
4,965,718	10/1990	George et al.	395/425
5,317,726	5/1994	Horst	395/575

OTHER PUBLICATIONS

Kinji Mori, et al., "Autonomous Decentralized Software Structure and Its Application," IEEE, Proceedings of the Fall Joint Computer Conference, Nov. 2–6, 1986, pp. 1056–1063.

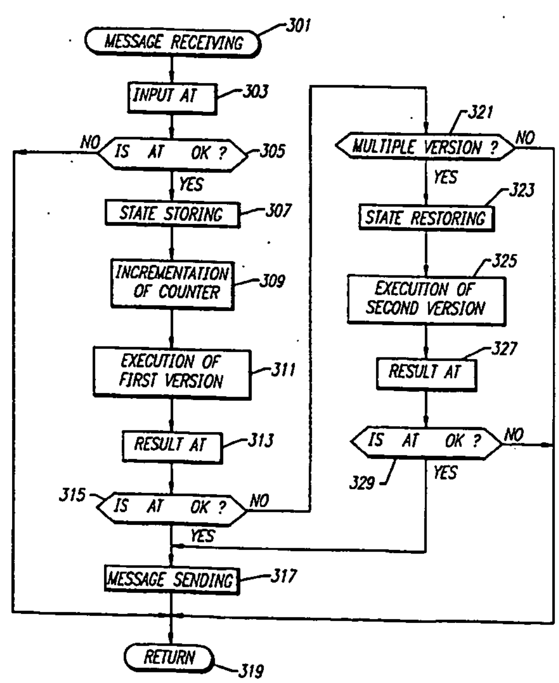
K. H. Kim, et al., "Distributed Execution of Recovery Blocks: an Approach for Uniform Treatment of Hardware and Software Faults in Real–Time Applications," *IEEE Transactions on Computers*, vol. 38, No. 5, May 1989, pp. 626–636.

Primary Examiner—Alyssa H. Bowler
Assistant Examiner—Alpesh M. Shah
Attorney, Agent, or Firm—Knobbe, Martens Olson & Bear

[57] **ABSTRACT**

A method for organizing and programming distributed computer systems in which processors are connected via interconnection or communication networks, such that even if many cases of hardware and software faults occur within the processors or within the networks, such faults do not lead to the failures in the application's computation. Parallel and asynchronous execution of multiple versions of a program module is performed with processors which are connected by a network without involving any direct interaction between the processors during the execution of the same or different versions of the program module. The process includes a step for executing, for each of the distributed program modules, the primary version and its backup version concurrently by use of multiple processors, a step for checking, in each processor, the logical acceptability of the output data produced from its execution of a program module version, a step for sending, in each processor, the acceptable output data to the transmission paths, a step for receiving, in each processor, the message from the transmission paths, checking the logical acceptability of each received message, and detecting the messages belonging to the same program module, and a step for selecting, in each processor, a message based on a selection logic.

18 Claims, 5 Drawing Sheets



11/2/2005, EAST Version: 2.0.1.4

received from the network, such as transmission line **27** of FIG. **1**, the logic flows to an "INPUT AT" block **303**, and then to a "IS AT OK?" block **305**. Blocks **303** and **305** in essence perform an input acceptance test for the received message. This input AT is the same as the test described in the first embodiment. In the event that AT is not OK, the logic proceeds to the end of the main sequence, as will be discussed later.

In the case where the result of the input AT is not OK, the message received is not used and is deleted. In the case where the result of the input AT is OK, the logic flows to a "STATE STORING" block **307**. In block **307**, the status information which is necessary for rollback to this point after the execution of the application module is saved in a buffer. Then, the logic moves to a "INCREMENTATION 0F COUNTER" block **309** which causes the counter areas in the module information table **165** of FIG. **3** that correspond to the application module to be executed to be incremented. The logic next moves to an "EXECUTION OF FIRST VERSION" block **311**. The logic of block **311** causes the first version module on the processors, namely A_p **71** on the processor **11** and Ab **73** on the processor **13**, to be executed. In a pair of blocks "RESULT AT" **313**, and "IS AT OK" **315**, the acceptance test is done for the output data of the respective modules A_p **71** and A_b **73**. This result AT is the same as the test described in the first embodiment. If the result of the test is OK, the logic moves to a "MESSAGE SENDING" block **317**, a message is prepared in the format in FIG. **2** and is sent to the network, including transmission line **27** of FIG. **1**, by way of the network interface module **151** as was shown in FIG. **3**. The logic then flows to a "RETURN" block **319**. "RETURN" block **319**, also receives a logic flow from the "IS AT OK?" block **305** in the event that the test for AT fails.

In the case where the result AT is not OK at block **315**, the logic flows to a "MULTIPLE VERSION?" block **321**, where an inquiry into whether or not there are multiple versions is made. If there are not multiple versions, the logic flows to the "RETURN" block **319**. If there are multiple versions, the logic flows to a "STATE RESTORING" block **323**, where the status before the execution of the first version module A_p **71** is recovered based upon the information that was stored during the execution of logic block **307**. The logic then proceeds to an "EXECUTION OF SECOND VERSION" block **325**, causing the second version module A_b **73** on the processor **11**, and A_p **71** on the processor **21**, to be executed. The logic than flows to a "RESULT AT" block **327** and an "IS AT OK" block **329**, in which the acceptance test is done for the output data of the respective modules A_p **71** in processor **13** and A_b **73** in processor **11**. If the acceptance test is met, the logic flows to the "MESSAGE SENDING" block **317**, and a message is prepared in the format in FIG. **2** and is sent to the network including transmission line **27** of FIG. **1**, by way of the network interface module **151** as was shown in FIG. **3**. If the acceptance test is not met, the logic flows to the "RETURN" block **319**.

However, in the case where the logic flows from "IS AT OK?" block **329** to a "MESSAGE SENDING" block **317** and a message is sent, the output message includes the information which indicates that the message occurred as the result of a retry, in addition to the formatting which was illustrated in FIG. **2**.

In the event that processor **11** crashes, the input AT **159** module of processors **19**, **21**, and **23** can recognize the crash of processor **11** by detecting the fact that the message **253** from the processor **11** is not received within the predetermined time. Then, the processor **23**, the processor which has been executing the primary version module X_p, sends a message, indicating the crash of processor **11**, to the processor **13**. And after receiving this message indicating the crash of processor **11**, processor **13** executes the module A_p **71** as the first version.

In addition, processor **23** may detect the situation where multiple producer processors, such as processors **11** and **13**, have executed the same version for processing the same data when the messages which have the same MI **109** part have been received. Messages which have the same MI **109** part have the same MN **111** part, the same AB **113** part and same SN **115** parts. In this case, the processor **23** sends the message indicating that both of the processors **11** and **13** have executed the same version. When the module A_p **71** has been used as the first version in both processors **11** and **13**, the processor **13** uses the module A_b **73** as the first version after receiving this message. When the module A_b **73** has been used as the first version in both processors **11** and **13**, the processor **11** will use the module A_p **71** as the first version after receiving this message. This mechanism is called the version switching suggestion (VSS) mechanism.

The utilization of this method, even if the primary processor which executes the primary version module as the first version crashes, enables assured execution of the primary version module. Moreover, a failure of a version in processing input data does not necessarily lead to the immediate dropout of a given processor from a distributed recovery block computing station. Again, a mechanism for status exchange among processors is not necessary in this method. The elimination of a requirement for an exchange mechanism under the inventive device and method herein provides flexibility and tolerance of both software and hardware faults.

Although the invention has been derived with reference to particular illustrative embodiments thereof, many changes and modifications of the invention may become apparent to those skilled in the art without departing from the spirit and scope of the invention. Therefore, included within the patent warranted hereon are all such changes and modifications as may reasonably and properly be included within the scope of this contribution to the art.

We claim:

1. A method for distributed redundant execution of program modules in a distributed system which has at least two processors connected by transmission medium and in which only one version of a program module is present and stored in each of at least two processors, said method comprising the steps of:

asynchronously executing, in each of said at least two processors, said program module to produce result data, after receiving an input data message containing input data necessary for the execution;

sending, by each of said at least two processors which executed said program module, to at least one destination processor an information message which contains at least both a result data produced by said execution of said program module and said input data utilized during said program module execution, through the transmission medium; and

receiving, in said at least one destination processor, said information messages and selecting one of the received information messages as a new input data message based upon both the result data and the input data contained in said received information messages.

2. A method for distributed redundant execution of program modules in a distributed system which has at least two

processors connected by transmission medium and in which only one version of a program module is present and stored in each one of at least two processors, said method comprising the steps of:

asynchronously executing, in each of said at least two processors, said program module to produce result data, after receiving an input data message containing input data necessary for this execution;

checking to conclusion, in each of said at least two processors which executed said program module, acceptability of the result data produced by said asynchronously executing said program module based only upon information within each of said at least two processors;

sending, by each of said at least two processors which executed said program module and completed said checking with positive conclusion about acceptability, to at least one destination processor an information message which contains at least the result data produced by said execution of said program module through the transmission medium; and

receiving, in said at least one destination processor, said information messages and selecting one of the received information messages as a new input data message by comparing the contents of the result data in said received information messages.

3. A method for distributed redundant execution of program modules in a distributed system which has at least two processors connected by transmission medium and in which only one version of a program module is present and stored in each one of at least two processors, said method comprising the steps of:

asynchronously executing, in each of said at least two processors, said program module to produce result data, after receiving an input data message containing input data necessary for said execution;

checking to conclusion, in each of said at least two processors which executed said program module, acceptability of the result data produced by said asynchronously executing said program module based only upon information within each of said at least two processors;

sending, by each of said at least two processors which executed said program module and completed said checking with a positive conclusion about the acceptability, to at least one destination processor an information message which contains at least both the result data produced by said execution of said program module and said input data utilized during said program module execution through the transmission medium; and

receiving, in said at least one destination processor, said information messages and selecting one of the received information messages as a new input data message based on both the result data and the input data contained in said received information messages.

4. A method for distributed redundant execution of program modules in a distributed system which has at least two processors connected by transmission medium and in which at least first and second different versions of a program module are stored in each one of at least two processors, said method comprising the steps of:

asynchronously executing, in each of said at least two processors, any of said at least first and second different versions of a program module which is stored therein to produce result data, after receiving an input data message containing input data necessary for said asynchronous execution;

sending, by each of said at least two processors which executed a version of the program module, to at least one destination processor an information message which contains at least both the result data produced by said asynchronous execution of said version and the information about said asynchronously executed version and about identity of said sending processor through the transmission medium;

receiving, in said at least one destination processor, said information messages and selecting an acceptable one of the received information messages as a new input data message; and

sending, from each destination processor which selected one of the received information messages, a second message which identifies which of said at least two processors produced said information messages that were discovered to be unacceptable during said selecting step, to the processors that produced said information messages.

5. The method for distributed redundant execution of program modules in a distributed system as recited in claim 4, wherein said second message identifies which, if any, of said at least two processors executed a same version of the program module.

6. The method for distributed redundant execution of program modules in a distributed system as recited in claim 4, wherein said selection of one of the received information messages is accomplished by comparing the contents of the result data in the received information messages.

7. The method for distributed redundant execution of program modules in a distributed system as recited in claim 5, wherein said selection of an acceptable one of the received information messages is accomplished by comparing the contents of the result data in the received information messages.

8. A method for distributed redundant execution of program modules in a distributed system which has at least two processors connected by transmission medium and in which at least first and second different versions of a program module are stored in each one of at least two processors, said method comprising the steps of:

asynchronously executing, in each of said at least two processors, any one of said at least first and second different versions of a program module which is stored therein to produce result data, after receiving an input data message containing input data necessary for said asynchronous execution;

sending, by each of said at least two processors, to at least one destination processor an information message which contains at least:

the result data produced by said execution of said one of said at least first and second different versions;

said input data utilized during said execution of said at least first and second different versions;

and information about said executed version through the transmission medium:

receiving, in said at least one destination processor, said information messages and selecting one of the information messages as a new input data message based on the result data and the input data contained in said information messages; and

sending, from said each destination processor which selected one of the received information messages, a second message which identifies which of said at least

two processors produced said information messages that were discovered to be unacceptable during said selecting step to the processor that produced said information message.

9. The method for distributed redundant execution of program modules in a distributed system as recited in claim **8**, wherein said second message identifies which of said at least two processors executed a same version of the program module.

10. A method for distributed redundant execution of program modules in a distributed system which has at least two processors connected by transmission medium and in which at least first and second different versions of a program module are stored in each of at least two processors, said method comprising the steps of:

asynchronously executing, in each of said at least two processors, any one of said at least first and second different versions of a program module to produce result data, after receiving an input data message containing input data necessary for said asynchronous execution;

checking to conclusion, in each of said at least two processors which executed said any one of said at least first and second different versions of the program module, acceptability of the result data produced by said execution based only upon information within each of said at least two processors;

sending, by each of at least two processors which executed one of said at least first and second different versions of the program module, to at least one destination processor an information message which contains at least:

the result data produced by said execution of said any one version and

the information about said asynchronously executed version and about identity of said sending processor through the transmission medium;

and

receiving, in said at least one destination processor, said information messages and selecting one of the information messages as a new input data message.

11. The method for distributed redundant execution of program modules in a distributed system as recited in claim **10**, wherein said selecting one of said information messages step is accomplished by selecting a first information message received.

12. The method for distributed redundant execution of program modules in a distributed system as recited in claim **10**, wherein said selecting one of said information messages step is accomplished based on the identities of said at least first and second versions of a program module that produced the information messages.

13. The method for distributed redundant execution of program modules in a distributed system as recited in claim **10**, wherein said selecting one of said information messages step is accomplished by comparing the contents of the result data in the received information messages.

14. The method for distributed redundant execution of program modules in a distributed system as recited in claim **13**, wherein each destination processor which selected one of the information messages sends a second message which identifies which of said at least two processors produced said information messages that were discovered to be unacceptable during said selecting step, to the processors that produced said information messages.

15. The method for distributed redundant execution of program modules in a distributed system as recited in claim **14**, wherein said second message identifies which, if any, of said at least two processors executed a same version of the program module.

16. A method for distributed redundant execution of program modules in a distributed system which has at least two processors connected by transmission medium and in which at least first and second different versions of a program module are stored in each of at least two processors, said method comprising the steps of:

asynchronously executing, in each of said at least two processors, any of said at least first and second different versions of a program module which is stored therein to produce result data, after receiving a message containing input data necessary for said asynchronous execution;

checking to conclusion, in each of said at least two processors which executed said one of said at least first and second different versions of the program module, acceptability of the result data produced by said asynchronously executing said one version of said program module based only upon information within each of said at least two processors;

sending, by each of at least two processors which executed one of said at least first and second different versions of the program module, to at least one destination processor an information message which contains at least:

the result data produced by said asynchronously executing said one version of said program module;

said input data utilized during said one version execution; and

information about said executed version and about identity of said sending processor through the transmission medium;

and

receiving, in said at least one destination processor, said information messages and selecting one of the information messages as a new input data message based on the result data and the input data contained in said received information messages.

17. The method for distributed redundant execution of program modules in a distributed system as recited in claim **16**, wherein each destination processor which selected one of the received messages sends a second message which identifies which of said at least two processors produced said information messages that were discovered to be unacceptable during said selecting step, to the processors that produced said information messages.

18. The method for distributed redundant execution of program modules in a distributed system as recited in claim **16**, wherein said second message identifies which, if any, said at least two processors executed a same version of the program module.

* * * * *

US005682470A

United States Patent [19]

Dwork et al.

[11] **Patent Number: 5,682,470**

[45] **Date of Patent: Oct. 28, 1997**

[54] **METHOD AND SYSTEM FOR ACHIEVING COLLECTIVE CONSISTENCY IN DETECTING FAILURES IN A DISTRIBUTED COMPUTING SYSTEM**

[75] Inventors: **Cynthia Dwork**, Palo Alto; **Ching-Tien Ho**; **Hovey Raymond Strong, Jr.**, both of San Jose, all of Calif.

[73] Assignee: **International Business Machines Corporation**, Armonk, N.Y.

[21] Appl. No.: **522,651**

[22] Filed: **Sep. 1, 1995**

[51] **Int. Cl.**[6] **G06F 11/00**

[52] **U.S. Cl.** **395/182.1**; 395/182.11; 395/182.02; 395/185.1; 395/200.11

[58] **Field of Search** 395/182.02, 182.09, 395/182.11, 185.01, 185.1, 183.01, 200.11, 200.03, 200.21, 184.01, 185.08, 182.1

[56] **References Cited**

U.S. PATENT DOCUMENTS

4,055,808	10/1977	Holsinger et al.	325/67
4,323,966	4/1982	Whiteside et al.	364/200
4,545,011	10/1985	Lyon et al.	395/183.19
4,569,015	2/1986	Dolev et al.	364/200
4,710,926	12/1987	Brown et al.	395/182.02
4,914,657	4/1990	Walter et al.	364/200
5,206,952	4/1993	Sundet et al.	395/725
5,365,512	11/1994	Combs et al.	395/182.02
5,377,322	12/1994	Ogura et al.	395/200
5,408,649	4/1995	Beshears et al.	395/600
5,436,909	7/1995	Dev et al.	371/20.1
5,485,465	1/1996	Liu et al.	395/182.02
5,506,955	4/1996	Chen et al.	395/184.01
5,513,354	4/1996	Dwork et al.	395/650
5,519,830	5/1996	Opoczynski	395/182.02

OTHER PUBLICATIONS

Vaidya et al., "Degradable Agreement in the Presence of Byzantine Faults", Distributed Computig syustems, 1993 Int'l Conf. IEEE, pp.237–244, 1993.

Becker, "Keeping Processes Under Surveillance", Reliable Distributed systems, 1991, 10th Symposium, IEEE, pp. 198–205.

Ng, "Ordered Broadcasts for Large Applications", Reliable Distributed Systems, 1991, 10th Symposium, IEEE, pp.188–197.

Yan et al., "Otimal Agreement Protocol in Malicious Faulty Processors and Faulty Links", IEEE Trans. on Knowledge and Data Eng., 1992, pp. 266–280.

Barborak et al., "Partitioning for Efficient Consensus" System Sciences, 1993 Annual Int'l Conf., IEEE, pp. 438–446, 1993.

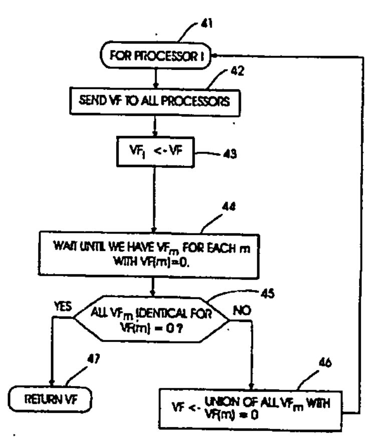
(List continued on next page.)

Primary Examiner—Robert W. Beausoliel, Jr.
Assistant Examiner—Joseph E. Palys
Attorney, Agent, or Firm—Khanh Q. Tran; James C. Pintner

[57] **ABSTRACT**

A method and apparatus are disclosed for achieving collective consistency in the detection and reporting of failures in a distributed computing system having multiple processors. Each processor is capable of being called by a parallel application for system status. Initially, each processor sends the other processors its view on the status of the processors. It then waits for similar views from other processors except those regarded as failed in its own view. If the received views are identical to the view of the processor, the processor returns its view to the parallel application. In a preferred embodiment, if the views are not identical to its view, the processor sets its view to the union of the received views and its current view. The steps are then repeated. Alternately, the steps are repeated if the processor does not have information that each of the processors not regarded as failed in its view forms an identical union view. In another preferred embodiment, the method is terminated if a quorum is not formed by the processors which are not regarded as failed. Alternatively, after sending its view, the processor waits for an exit condition. Depending on the exit condition, the processor sets its view to a quorum view and sends a "DECIDE" message to the other processors. In another embodiment, the processor updates its view and the method steps are repeated.

47 Claims, 9 Drawing Sheets



"DECIDE" message from one of the other processors, and the "DECIDE" message includes a view VF which contains the view VF of processor. In this case, the view VF of processor i is set to the view VF according to the method step **70**. The processor i then sends a special "DECIDE" message with VF to all other processors and returns from the message interface **9** with its view according steps **68** and **47**, respectively, similar to the first exit condition.

Still referring the flowchart depicted in FIG. **9**, block **73** represents the third exit condition from the waiting step **65** of the method. This condition occurs when the processor i has sufficient information to determine that there is only one view VF that any processor could possibly have received from a quorum of processors and VF contains the view of the processor i. According to step **74** of the method, the view VF of processor i is set to be equal to the view VF. Next, the method continues by repeating the method steps starting with step **41**. As an example, assuming the adopted quorum family is the family of majority sets of processors, the processor i would exit step **65** via the condition block **73** if it has received views VF from exactly half the processors participating in the protocol so that no processor would receive a different view from a quorum of the processors. Next, the view of processor i is updated with view VF. The method steps are then iterated beginning with step **41**.

The fourth exit condition is shown by block **75** of the flowchart of FIG. **9**. The condition holds true when the processor i has sufficient information to determine that no participant processor has received identical views from a quorum of processors. In this case, the view VF of the processor i is updated to be the same as the union of all views it received, according to step **76**. As a result, the set of failed processors reported in VF is the union of the sets of failed processors reported in all the views received by processor i. The method steps are next repeated starting with step **41**, as in the case of the third exit condition. Thus, in step **42**, processor i sends its updated view VF before checking again for one of the exit conditions in block **65**. For example, if the adopted quorum family is the family of majority sets of processors, processor i would exit the waiting step **65** via condition block **75** if processor i had received pairwise distinct views from two more than half the processors.

FIG. **10** illustrates in the form of a flowchart, a preferred embodiment for a method for detecting failures in a distributed system to be practiced with the method for achieving collective consistency of the present invention. The embodiment is referred to as a Multiphase Packet Oriented Failure Detector which may be used as a failure detector **32** shown in the flowchart of FIG. **4**. Again, the notation Vx(m) is used to indicate the value corresponding to processor from the data structure Vx where Vx ranges over the data structures VR, VN, VT, VI, and VF.

The method steps of the failure detector are performed logically concurrently for each processor **3** of the distributed computing system **1**. In summary, the method includes the first step of determining by each processor **3** a pattern of failures by all the processors to meet certain deadlines. In the second step, a local view is established by the processor **3** as to which of the processors have failed, using the determined pattern of failures. Next, the processor **3** exchanges its local view with the views similarly formed by the other processors. The exchange is based on a collective consistency protocol which assures that if any two processors **3** return their views and one processor is not regarded as failed in the view of the other processor, then the two returned views are identical. As a result of performing these method steps, failed processors **3** are consistently detected and reported by all the processors.

In the preferred embodiment illustrated by the flowchart of FIG. **10**, the method steps are executed logically concurrently by each processor i for each processor or for which processor i has VT(m)>0. The notation VT(m) represents the total number of data packets to be received by processor i from each of the other processors m. In step **81**, a nonblocking multireceive call with input parameter VR is issued. Accordingly, processor/waits until all data packets VR(m) from each processor or are received. In step **82**, the method determines whether a certain condition is satisfied before processor i exits the waiting step **81**. In a preferred embodiment of the invention, the exit condition is whether all VR(m) packets are received by the processor i before a timeout. In the case of the affirmative branch from step **82**, i.e., when VN(m)=0, the corresponding failure indicator VI(m) is set to zero (0), according to step **83**. Next, in step **84**, the number of packets to be received VT(m) is decremented by the number of requested packets actually received by processor i from processor or, i.e., by the value of VR(m)–VN(m). The method continues with step **80** to begin another round of receiving packets.

In the case of the negative branch from the decision block **82**, the failure indicator VI(m) corresponding to processor m is incremented by VN(m) according to step **85**. The value of VI(m) is then compared with a threshold for processor i to receive packets from the other processors. In a preferred embodiment, the interface **2** includes a broadcast medium and the threshold is for processor i to receive packets sent to the broadcast medium by the other processors m.

If the failure indicator VI(m) exceeds the threshold, processor i concludes that processor m has either failed or is too slow in its operation. The processor i then sets its view VF(m) to one (1) to indicate that processor or has failed and sets VT(m) to zero (0) to indicate that no further packets are to be requested from processor m. The method continues with step **80** to begin another round of sending and receiving packets. In the case the failure indicator VI(m) does not exceed the threshold, step **84** is performed and the method similarly continues with step **80**.

In another preferred embodiment of the method for detecting faults illustrated by flowchart of FIG. **4** and based on the failure detector of FIG. **10**, the method further includes the step of reorganizing remaining work to be performed by the distributed system **1** among the still operational processors **3** once failed processors are detected.

While several preferred embodiments of the invention have been described, it should be apparent that modifications and adaptations to those embodiments may occur to persons skilled in the art without departing from the scope and the spirit of the present invention as set forth in the following claims.

What is claimed is:

1. A computer-implemented method for achieving collective consistency in detecting failures in a distributed computing system, the system having a plurality of processors participating in a collective call by a parallel application, the method comprising the steps, performed for each processor, of:

sending, by the processor to the other processors, a view representing the status of the other processors, including status as to which of the other processors have failed;

waiting until the processor receives views from all other processors except those regarded as failed in the view of the processor, the views being sent by execution of the step of sending; and

returning the view of the processor to the parallel application if the received views are identical to the view of the processor,

whereby collective consistency is achieved, collective consistency being a condition which holds whenever it is true that, if any two processors return their views to the parallel application and one of the two processors is not regarded as failed in the view of the other processor, then the two views are identical.

2. The method as recited in claim 1 further comprising the steps, performed immediately before the step of returning if the received views are not identical to the view of the processor, of:

updating the view of the processor to be the same as a union of all the received views and the current view of the processor; and

repeating the steps of the method.

3. The method as recited in claim 1 further comprising the steps, performed immediately before the step of returning, of:

updating the view of the processor to be the same as a union of all the received views and the current view of the processor;

determining whether the processor has information that each of the other processors not regarded as failed in the union forms a union identical to the union of the processor; and

repeating the steps of the method.

4. The method as recited in claim 2 further comprising the steps, performed immediately after the step of sending, of:

determining whether the processors not regarded as failed in the view of the processor form a quorum, the quorum being any element of a previously chosen plurality of subsets of a set, the subsets having a property that any two of the subsets intersect; and terminating the method without returning to the parallel application if the quorum is not formed.

5. The method as recited in claim 1, wherein

the step of waiting is performed until:

a group of the processors form a quorum, the quorum being any element of a previously chosen plurality of subsets of a set, the subsets having a property that any two of the subsets intersect;

the views received from the processors in the quorum are identical; and

each of the identical views contains the view of the processor; and

the method further comprises the steps, performed immediately before the step of returning, of:

updating the view of the processor to be the same as the identical views; and

sending a DECIDE message to all other processors, the message including the updated view of the processor.

6. The method as recited in claim 1, wherein

the step of waiting is performed until:

the processor receives a DECIDE message from one of the other processors; and

the message has a view which includes the current view of the processor; and

the method further comprises the steps, performed immediately after the step of waiting, of:

updating the view of the processor to be the same as the view of the message; and

sending a DECIDE message to all other processors, the message including the updated view of the processor.

7. The method as recited in claim 1, wherein

the step of waiting is performed until:

the processor has information of a second view that contains the view of the processor, and

the second view is the only view that could possibly be held by a quorum, the quorum being any element of a previously chosen plurality of subsets of a set, the subsets having a property that any two of the subsets intersect; and

the method further comprises the steps, executed immediately after the step of waiting, of:

updating the view of the processor to be the same as the second view; and

repeating the steps of the method.

8. The method as recited in claim 1, wherein

the step of waiting is performed until:

the processor has information that none of the other processors received identical views from a group of the processors, the group forming a quorum, the quorum being any element of a previously chosen plurality of subsets of a set, the subsets having a property that any two of the subsets intersect; and

the method further comprises the steps, performed immediately after the step of waiting, of:

updating the view of the processor to be the same as a union of the views received and the current view of the processor; and

repeating the steps of the method.

9. A computer-implemented method for detecting and reporting failures among a plurality of processors of a distributed computing system, the processors participating in a collective call by a parallel application, the method comprising the steps of:

determining, by each processor of the call, a pattern of failures by the processors to meet a plurality of deadlines;

forming a local view, by the processor, as to which ones of the processors have failed, the local view being based on the pattern; and

exchanging the local view with the views of the other processors, the exchange being based on a collective consistency protocol such that if any two processors return their views to the parallel application and one of the two processors is not regarded as failed in the view of the other processor, then the two views are identical, whereby failed ones of the processors are consistently detected and reported by the processors.

10. The method as recited in claim 9, wherein the collective consistency protocol is such that if any two processors participating in a collective call return their views to the parallel application, then the views returned are identical.

11. The method as recited in claim 9, wherein the collective consistency protocol is such that if any failure is detected locally by a processor participating in a collective call and the processor returns to the parallel application, then the failure is reported to the interface.

12. The method as recited in claim 9, wherein the interface includes a broadcast medium and the deadlines are for the processors to receive a plurality of packets from the broadcast medium, the packets being sent to the broadcast medium by the processors.

13. The method as recited in claim 12, wherein the step of determining includes the step of associating with each pattern a numerical measure relative to a threshold for receiving the packets by the processors.

14. The method as recited in claim 13, wherein the numerical measure is a number of consecutive failures by a processor to meet a deadline.

15. The method as recited in claim 9 further comprising the step of reorganizing work to be performed by the distributed computing system among the processors not detected as failed.

16. A processor for use in a distributed computing system of processors, the processors participating in a collective call by a parallel application for status of the distributed system, the processor comprising:

means for sending to the other processors of the distributed system a view of the processor on the status of the other processors, including status as to which of the other processors have failed;

means for waiting until the processor receives the views from all other processors except those regarded as failed in the view of the processor, the views being sent by execution of the step of sending; and

means for returning the view of the processor to the parallel application if the received views are identical to the view of the processor,

whereby collective consistency is achieved among the processors as to the status of the processors returned to the parallel application, collective consistency being a condition which holds whenever it is true that, if any two processors return their views to the parallel application and one of the two processors is not regarded as failed in the view of the other processor, then the two views are identical.

17. The processor as recited in claim 16 further comprising

means for updating the view of the processor to be the same as a union of all the views received and the current view of the processor, if the received views are not identical to the view of the processor.

18. The processor as recited in claim 16 further comprising:

means for updating the view of the processor to be the same as a union of all the views received and the current view of the processor;

means for determining whether the processor has information that each of the other processors not regarded as failed in the union forms a union identical to the union of the processor; and

means for repeating the operation of the processor if the processor does not have the information.

19. The processor as recited in claim 17 further comprising:

means for determining whether the processors not regarded as failed in the view of the processor form a quorum, the quorum being any element of a previously chosen plurality of subsets of a set, the subsets having a property that any two of the subsets intersect; and

means for terminating without returning to the parallel application if the quorum is not formed, the means for determining the quorum and terminating means operating immediately after the operation of the means for sending.

20. The processor as recited in claim 16, wherein

the waiting means operates until:

a group of the processors form a quorum, the quorum being any element of a previously chosen plurality of subsets of a set, the subsets having a property that any two of the subsets intersect;

the views received from the processors in the quorum are identical; and

each of the identical views contains the view of the processor; and

the processor further comprises:

means for updating the view of the processor to be the same as the identical views; and

means for sending a DECIDE message to all other processors, the message including the updated view of the processor.

21. The processor as recited in claim 16, wherein

the means for waiting operates until:

the processor receives a DECIDE message from one of the other processors; and

the message has a view which includes the current view of the processor; and

the processor further comprises:

means for updating the view of the processor to be the same as the view of the message; and

means for sending a DECIDE message to all other processors, the message including the updated view of the processor.

22. The processor as recited in claim 16, wherein

the means for waiting operates until:

the processor has information of a second view that contains the view of the processor, and

the second view is the only view that could possibly be held by a quorum, the quorum being any element of a previously chosen plurality of subsets of a set, the subsets having a property that any two of the subsets intersect; and

the processor further comprises:

means for updating the view of the processor to be the same as the second view; and

means for repeating the operation of the processor.

23. The processor as recited in claim 16, wherein

the waiting means operates until:

the processor has information that none of the other processors received identical views from a group of the processors, the group forming a quorum, the quorum being any element of a previously chosen plurality of subsets of a set, the subsets having a property that any two of the subsets intersect; and

the processor further comprises:

means for updating the view of the processor to be the same as a union of the views received and the current view of the processor; and

means for repeating the operation of the processor.

24. A processor for use in a distributed computing system of processors, the processors participating in a collective call by a parallel application, the processor comprising:

mean for determining a pattern of failures by the processors to meet a plurality of deadlines;

means for forming a local view as to which ones of the processors have failed, the local view being based on the pattern; and

means for exchanging the local view with the views of the other processors, the exchanging means operating according to a collective consistency protocol such that if any two processors participating in the collective call return their views to the parallel application and one of the two processors is not regarded as failed in the view of the other processor, then the two views are identical, whereby failed ones of the processors are consistently detected and reported by the processors.

25. The processor as recited in claim 24, wherein the collective consistency protocol is such that if any two processors participating in the collective call return their views to the parallel application, then the views returned are identical.

26. The processor as recited in claim 24, wherein the collective consistency protocol is such that if any failure is detected locally by a second processor participating in the collective call and the second processor returns to the parallel application, then the failure is reported to the interface by the processor.

27. The processor as recited in claim 24, wherein the interface includes a broadcast medium and the deadlines are for the processors to receive a plurality of packets from the broadcast medium, the packets being sent to the broadcast medium by the processors.

28. The processor as recited in claim 27, wherein the means for determining includes means for associating with the pattern a numerical measure relative to a threshold for receiving the packets by the processors.

29. The processor as recited in claim 28, wherein the numerical measure is a number of consecutive failures by the processors to meet a deadline.

30. A distributed computing system having a plurality of processors participating in a collective call by a parallel application for status of the system, each processor comprising:

means for sending to the other processors of the system a view on the status of the processors, including status as to which of the other processors have failed;

means for waiting until the processor receives views from all other processors except those regarded as failed in the view of the processor, the views being sent by execution of the step of sending; and

means for returning the view of the processor to the parallel application if the received views are identical to the view of the processor,

whereby collective consistency is achieved among the processors as to the status of the processors returned to the parallel application, collective consistency being a condition which holds whenever it is true that, if any two processors return their views to the parallel application and one of the two processors is not regarded as failed in the view of the other processor, then the two views are identical.

31. A distributed computing system having a plurality of processors participating in a collective call by a parallel application each processor comprising:

mean for determining a pattern of failures by the processors to meet a plurality of deadlines;

means for forming a local view as to which ones of the processors have failed, the local view being based on the pattern; and

means for exchanging the local view with the views of the other processors, the exchanging means operating according to a collective consistency protocol such that if any two processors return their views to the parallel application and one of the two processors is not regarded as failed in the view of the other processor, then the two views are identical, whereby failed ones of the processors are consistently detected and reported by the processors.

32. The distributed computing system as recited in claim 31 further comprising means for reorganizing work to be performed by the distributed computing system among the processors not detected as failed.

33. A computer program product for achieving collective consistency in detecting failures in a distributed computing system, the system having a plurality of processors participating in a collective call by a parallel application for status on the processors, the computer program product comprising:

a recording medium;

means, recorded on the recording medium, for instructing each of the processors to perform the steps of:

sending to the other processors a view representing the status of the other processors, including status as to which of the other processors have failed;

waiting until the processor receives the views from all other processors except those regarded as failed in the view of the processor, the views being sent by execution of the step of sending; and

returning the view of the processor to the parallel application if the received views are identical to the view of the processor,

whereby collective consistency is achieved among the processors as to the status of the processors returned to the parallel application, collective consistency being a condition which holds whenever it is true that, if any two processors return their views to the parallel application and one of the two processors is not regarded as failed in the view of the other processor, then the two views are identical.

34. The computer program product as recited in claim 33 further comprising means, recorded on the recording medium, for instructing the processor to perform, immediately before the step of returning if the received views are not identical to the view of the processor, the steps of:

updating the view of the processor to be the same as a union of all the received views and the current view of the processor; and

repeating the steps of the program product.

35. The computer program product as recited in claim 33 further comprising means, recorded on the recording medium, for instructing the processor to perform, immediately before the step of returning, the steps of:

updating the view of the processor to be the same as a union of all the views received and the current view of the processor;

determining whether the processor has information that each of the other processors not regarded as failed in the union forms a union identical to the union of the processor; and

repeating the steps of the program product if the processor does not have the information.

36. The computer program product as recited in claim 34 further comprising means, recorded on the recording medium, for instructing the processor to perform, immediately after the step of sending, the steps of:

determining whether the processors not regarded as failed in the view of the processor form a quorum, the quorum being any element of a previously chosen plurality of subsets of a set, the plurality of subsets having a property that any two elements of the plurality of subsets intersect; and

terminating the method without returning to the parallel application if the quorum is not formed.

37. The computer program product as recited in claim 33, wherein

the step of waiting is performed until:

a group of the processors form a quorum, the quorum being any element of a previously chosen plurality of subsets of a set, the subsets having a property that any two of the subsets intersect;

the views received from the processors in the quorum are identical; and

each of the identical views contains the view of the processor; and

the computer program product further comprises means, recorded on the recording medium, for instructing the processor to perform, immediately before the step of returning, the steps of:

updating the view of the processor to be the same as the identical views; and

sending a DECIDE message to all other processors, the message including the updated view of the processor.

38. The computer program product as recited in claim **33**, wherein

the step of waiting is performed until:

the processor receives a DECIDE message from one of the other processors; and

the message has a view which includes the current view of the processor; and

the computer program product further comprises means, recorded on the recording medium, for instructing the processor to perform, immediately after the step of waiting, the steps of:

updating the view of the processor to be the same as the view of the message; and

sending a DECIDE message to all other processors, the message including the updated view of the processor.

39. The computer program product as recited in claim **33**, wherein

the step of waiting is performed until:

the processor has information of a second view that contains the view of the processor, and

the second view is the only view that could possibly be held by a quorum, the quorum being any element of a previously chosen plurality of subsets of a set, the plurality of subsets having a property that any two elements of the plurality of subsets intersect; and

the computer program product further comprises means, recorded on the recording medium, for instructing the processor to perform, immediately after the step of waiting, the steps of:

updating the view of processor to be the same as the second view; and

repeating the steps of the program product.

40. The computer program product as recited in claim **33**, wherein

the step of waiting is performed until:

the processor has information that none of the other processors received identical views from a group of the processors, the group forming a quorum, the quorum being any element of a previously chosen plurality of subsets of a set, the subsets having a property that any two of the subsets intersect; and

the computer program product further comprises means, recorded on the recording medium, for instructing the processor to perform, immediately after the step of waiting, the steps of:

updating the view of the processor to be the same as a union of the views received and the current view of the processor; and

repeating the steps of the program product.

41. A computer program product for detecting and reporting failures in a distributed computing system, the system having a plurality of processors participating in a collective call by a parallel application, the computer program product comprising:

a recording medium;

means, recorded on the recording medium, for instructing each of the processors to perform the steps of:

determining a pattern of failures by the processors to meet a plurality of deadlines;

forming a local view as to which ones of the processors have failed, the local view being based on the pattern; and

exchanging the local view with the views of the other processors, the exchange being based on a collective consistency protocol such that if any two processors return their views to the parallel application and one of the two processors is not regarded as failed in the view of the other processor, then the two views are identical, whereby failed ones of the processors are consistently detected and reported by the processors.

42. The computer program product as recited in claim **41**, wherein the collective consistency protocol is such that if any two processors participating in the collective call return their views to the parallel application, then the two views returned are identical.

43. The computer program product as recited in claim **41**, wherein the collective consistency protocol is such that if any failure is detected locally by a processor participating in the collective call and the processor returns to the parallel application, then the failure is reported to the interface.

44. The computer program product as recited in claim **41**, wherein the interface includes a broadcast medium and the deadlines are for the processors to receive a plurality of packets from the broadcast medium, the packets being sent to the broadcast medium by the processors.

45. The computer program product as recited in claim **44**, wherein the means for instructing the processor to perform the step of determining includes means, recorded on the recording medium, for instructing the processor to perform the step of associating with each pattern a numerical measure relative to a threshold for receiving the packets by the processors.

46. The computer program product as recited in claim **45**, wherein the numerical measure is a number of consecutive failures by a processor to meet a deadline.

47. The computer program product as recited in claim **41** further comprising means, recorded on the recording medium, for instructing the distributed computing system to reorganize work to be performed by the distributed system among the processors not detected as failed.

* * * * *

US006279104B1

(12) United States Patent
Sato et al.

(10) Patent No.: **US 6,279,104 B1**
(45) Date of Patent: ***Aug. 21, 2001**

(54) **DEBUGGING SYSTEM FOR PARALLEL PROCESSED PROGRAM AND DEBUGGING METHOD THEREOF**

(75) Inventors: **Yuji Sato; Norihisa Murayama**, both of Shizuoka (JP)

(73) Assignee: **Fujitsu Limited**, Kawasaki (JP)

(*) Notice: This patent issued on a continued prosecution application filed under 37 CFR 1.53(d), and is subject to the twenty year patent term provisions of 35 U.S.C. 154(a)(2).

Subject to any disclaimer, the term of this patent is extended or adjusted under 35 U.S.C. 154(b) by 0 days.

(21) Appl. No.: **09/047,952**

(22) Filed: **Mar. 26, 1998**

(30) **Foreign Application Priority Data**

Jul. 22, 1997 (JP) 9-196237

(51) **Int. Cl.**[7] **G06F 11/00**

(52) **U.S. Cl.** **712/227**; 712/228; 712/229

(58) **Field of Search** 395/704; 345/440; 709/102, 300; 712/227, 228, 229

(56) **References Cited**

U.S. PATENT DOCUMENTS

5,179,702 *	1/1993	Spix et al.	709/102
5,325,530 *	6/1994	Mohrmann	395/704
5,640,500 *	6/1997	Taylor	345/440
5,687,375 *	11/1997	Schwiegelshohn	395/704
5,745,760 *	4/1998	Kawamura et al.	709/300
5,963,746 *	10/1999	Barker et al.	712/20

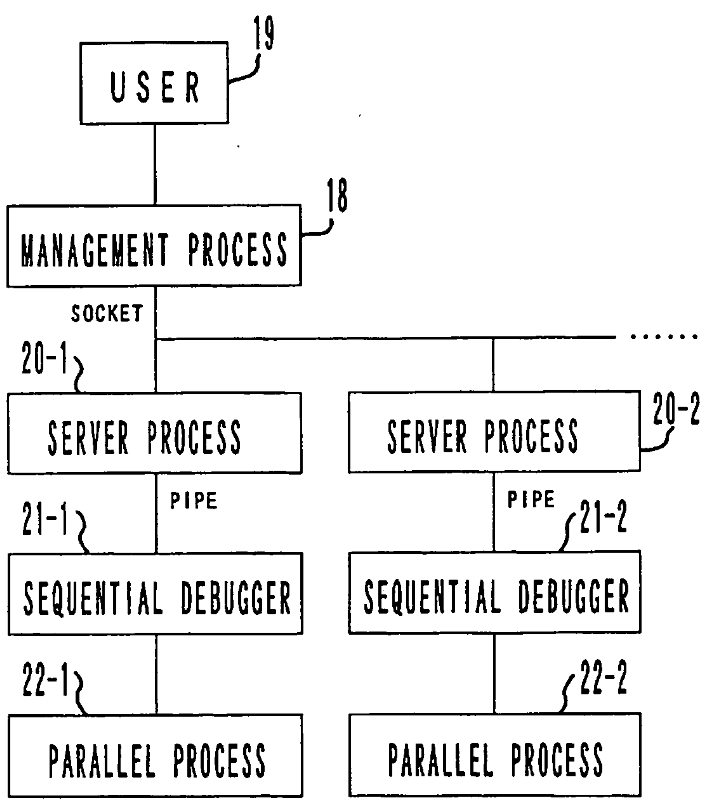
* cited by examiner

Primary Examiner—Reba I. Elmore
(74) *Attorney, Agent, or Firm*—Staas & Halsey LLP

(57) **ABSTRACT**

A debugging system for use with a data parallel processing apparatus is disclosed. Sequential debuggers debug a plurality of parallel processes. The processed result is output as reply information to a management processor. The management processor knows the reason why it has received reply information and manages debugging statuses of the individual sequential debuggers corresponding to the reply information against the debug command.

35 Claims, 25 Drawing Sheets



DEBUGGING SYSTEM FOR PARALLEL PROCESSED PROGRAM AND DEBUGGING METHOD THEREOF

BACKGROUND OF THE INVENTION

1. Field of the Invention

The present invention relates to a data parallel processing apparatus for processing data with a plurality of processor elements. More particularly, the present invention relates to a debugging system for a parallel processed program that drives the processing apparatus and a debugging method thereof.

2. Description of the Related Art

In a scientific and engineering parallel computer that repeats the same calculating process or the calculations with varied parameters, a data parallel process of which the calculating process is divided by a plurality of processors is performed. In the data parallel processing apparatus, since the same calculation program is executed by a plurality of processor elements in parallel, they perform respective calculating processes with respective data and variables so as to accomplish a high speed calculating process.

On the other hand, in the final stage of software development, a debugging process for checking out source code, executable format, and so forth and for finding errors of a program and of data, and variables, in the program is essential. Conventionally, the debugging process is performed by, for example, the following debugger.

The debugger debugs programs of individual processor units. This type of debugger is referred to as a sequential debugger. The sequential debugger debugs only a process program in one processor element. The debugger tracks the operation of a program of each processor element, stops for example a source program at a particular line, then checks out data and content of variables.

However, when a program for the data parallel processing apparatus with a plurality of processor elements connected in parallel is debugged, a complicated debugging process should be performed for each processor element. Thus, the debugging process is ineffective and thereby takes a long time.

SUMMARY OF THE INVENTION

An object of the present invention is to provide a debugging system for effectively performing a debugging process for a parallel processed program of a data parallel processing apparatus, and thereby largely contributing supporting program development as an effective debugging tool of a parallel processed program and a debugging method thereof.

The present invention is accomplished by a debugging system for use with a data parallel processing apparatus having a plurality of processor element for processing the same process in parallel, the debugging system comprising a plurality of sequential debuggers for debugging process programs of the processor elements, and a management processor for outputting a debug command to the sequential debuggers, causing the sequential debuggers to debug the process programs, receiving reply information therefrom as the debugged results, and managing a debugging process for the data parallel processing apparatus.

The plurality of processor elements (sometimes referred to as PEs) process the same program in parallel. For example, the processor elements process a parallel program in the scientific and engineering calculating process. Each processor element can access a shared memory shared by

each processor element along with a dedicated local memory therof. For example, each processor can access the shared memory along with a relevant dedicated local memory.

Each sequential debugger debugs only a program process of a relevant processor element. The sequential debugger performs a debugging process corresponding to a debug command that is received from the management processor. Examples of the debug command are "PRINT" command, "BREAK" command, "CONTINUE" command, "STEP" command, and "P.STAT" command.

In addition, the management processor outputs a debug command to each processor element. The sequential debugger of each processor element debugs the process program corresponding to each processor element. The management processor receives reply information from each processor element as a result of the debugging process for data of the relevant processor element.

Thus, the management processor outputs various debug commands to the sequential debuggers of the individual processor elements and receives reply information from the sequential debuggers. Consequently, the management processor can obtain the statuses of the programs of all the processor elements and effectively manage the debugging process. Thus, the management processor can effectively perform the debugging process.

These and other objects, features and advantages of the present invention will become more apparent in light of the following detailed description of a best mode embodiment thereof, as illustrated in the accompanying drawings.

BRIEF DESCRIPTION OF THE DRAWINGS

FIG. 1 is a schematic diagram showing the structure of the data parallel processing apparatus for which the debugging system according to the present invention is applied;

FIG. 2 is a schematic diagram for explaining a management processor;

FIG. 3 is a block diagram showing the structure of a debugging system according to the present invention, the debugging system being applied for the data parallel processing apparatus;

FIG. 4 is a schematic diagram showing the structure of processor elements including sequential debuggers and parallel processors;

FIG. 5 is a schematic diagram showing display examples of windows displayed on a display of the management processor;

FIG. 6 is a schematic diagram showing a display structure in the case that only source programs are displayed on the display;

FIG. 7 is a flow chart showing a process of the management processor;

FIG. 8 is a flow chart for explaining a debug command sending process performed by the management processor;

FIG. 9 is a flow chart for explaining a process performed by a sequential debugger;

FIG. 10 is a flow chart for explaining a reply information receiving process performed by the management processor;

FIG. 11 is a flow chart showing a transmission function processing program;

FIG. 12 is a flow chart for explaining addition of identification information performed by a server processor;

FIGS. 13A and 13B are schematic diagrams showing data in a status display portion after a "PRINT" command is output;

"breaks" continue as shown in FIG. 21, one current process is designated and thereby only "1: break" may be displayed. Thus, the other "breaks" may be suppressed from being displayed. However, in the example shown in FIG. 21, the process should be performed after reply information of "breaks" of all the sequential debuggers has been confirmed.

In the above-described embodiment, four debuggers were used as sequential debuggers. However, as the number of parallel processes of the parallel processing apparatus increases, the number of sequential debuggers increases.

FIG. 25 is a schematic diagram showing a storing medium that stores a program with which the embodiment is accomplished.

As described above, according to the present invention, the following effects can be accomplished.

When parallel processed programs of a data parallel processing apparatus are debugged, since the user can use a management processor that manages each sequential debugger, the debugging process can be very effectively performed.

In addition, corresponding to the function of a debug command, since a plurality of processes can be performed at a time, the debugging process can be very quickly performed.

Although the present invention has been shown and described with respect to a best mode embodiment thereof, it should be understood by those skilled in the art that the foregoing and various other changes, omissions, and additions in the form and detail thereof may be made therein without departing from the spirit and scope of the present invention.

What is claimed is:

1. A debugging system for use with a data parallel processing apparatus having a plurality of processor elements for processing the same process in parallel, the debugging system comprising:

a plurality of sequential debuggers to debug process programs of the processor elements, wherein each processor element has a local memory to store data peculiar to each said processor element and a common memory to store data common to all processor elements such that during execution of the respective debug process programs, each processor element outputs reply information in response to receipt of a debug command; and

a management processor to output the debug command to said sequential debuggers, to cause said sequential debuggers to debug the process programs, to receive the reply information therefrom as debugged results, and to add identification to the reply information representing that the reply information from each respective sequential debugger has been received.

2. The debugging system as set forth in claim 1, further comprising:

a server processor to control a corresponding sequential debugger at an instruction of said management processor, wherein said management processor outputs the debug command to said sequential debuggers through said server processor and to receive reply information from said sequential debuggers through the server processor, data between said management processor and the server processor being communicated on a socket communication basis, data between the server processor and said sequential debuggers being communicated on a pipe communication basis.

3. The debugging system as set forth in claim 1,

wherein said management processor does not output the next debug command until said management processor receives reply information from all of said sequential debuggers.

4. The debugging system as set forth in claim 1,

wherein said management processor displays a status when reply information is not received from all of said sequential debuggers.

5. The debugging system as set forth in claim 1,

wherein the debug command uses a transmission function contained in a source program using variables stored in global memory.

6. The debugging system as set forth in claim 1,

wherein the debug command is a status display command.

7. The debugging system as set forth in claim 1,

wherein the debug command is output to a selected one of said sequential debuggers.

8. The debugging system as set forth in claim 7,

wherein said management processor selects said sequential debuggers.

9. The debugging system as set forth in claim 1,

wherein the debug command is output to all of said sequential debuggers.

10. The debugging system as set forth in claim 9,

wherein said management processor selects said sequential debuggers.

11. The debugging system as set forth in claim 1,

wherein the reply information is displayed in a basic window of a display of said management processor.

12. The debugging system as set forth in claim 11,

wherein a process window for displaying the same source program of the processor elements or displaying different source programs of the processor element in parallel is generated on the display.

13. The debugging system as set forth in claim 1,

wherein the status display is displayed in a basic window of a display of said management processor.

14. The debugging system as set forth in claim 13,

wherein a process window for displaying the same source program of the processor elements or displaying different source programs of the processor element in parallel is generated on the display.

15. The debugging system as set forth in claim 1,

wherein the reply information for each of said sequential debuggers is grouped and displayed.

16. The debugging system as set forth in claim 15,

wherein the grouped reply information is displayed in an order of priority previously specified.

17. The debugging system as set forth in claim 15,

wherein the reply information is not displayed when the reply information is redundant information.

18. A debugging management processor for use with a sequential debugger for a data parallel processing apparatus having a plurality of processor elements for processing the same process in parallel, wherein each processor element has a local memory to store data peculiar to each said processor element and a common memory to store data common to all processor elements, the debugging management processor comprising means for outputting a debug

command to said a sequential debugger for debugging process programs of the processor elements, means for causing the sequential debugger to debug the process programs, means for receiving reply information as the debugged result from the sequential debugger, and means for managing a debugging process of the data parallel processing apparatus by adding identification to the reply information representing that the reply information from each respective sequential debugger has been received.

19. A readable storage medium for storing a program for causing a computer to perform:

(a) outputting a debug command to a process element for processing the same parallel program, wherein the process element has a local memory to store data peculiar to the process element and a common memory to store data common to a plurality of process elements; and

(b) receiving reply information of which a data parallel processed program of the process element has been debugged corresponding to the debug command that has been output by the operation (a) and managing a debugging process of the data parallel processed program by adding identification to the reply information representing that the reply information from the process element has been received.

20. A debugging method of a data parallel processed program of which a plurality of process elements process the same processes in parallel, comprising:

(a) outputting a debug command to the process elements, wherein each process element has a local memory to store data peculiar to each said process element and a common memory to store data common to all process elements;

(b) debugging data parallel processed programs of the process elements corresponding to the debug command that has been output at the operation (a); and

(c) receiving reply information as a result at the operation (b) and managing a debugging process of the data parallel processed program by adding identification to the reply information representing that the reply information from each respective process element has been received.

21. The debugging method as set forth in claim 20, wherein the debug command is a status display command.

22. The debugging method as set forth in claim 21, wherein the debug command uses a transmission function contained in a source program.

23. The debugging method as set forth in claim 20, wherein the operation (a) and the receiving of the operation (c) are performed corresponding to a server process.

24. The debugging method as set forth in claim 23, wherein the debug command is a status display command.

25. The debugging method as set forth in claim 24, wherein the reply information is displayed in a basic window of a display of the management processor.

26. The debugging method as set forth in claim 24, wherein the status display is displayed in a basic window of a display of the management processor.

27. The debugging method as set forth in claim 26, wherein a process window for displaying the same source program of the processor elements or displaying different source programs of the processor element in parallel is generated on the displayprocessor.

28. The debugging method as set forth in claim 23, wherein the debug command uses a transmission function contained in a source program.

29. The debugging method as set forth in claim 28, wherein the reply information is displayed in a basic window of a display of the management.

30. The debugging method as set forth in claim 28, wherein the status display is displayed in a basic window of a display of the management processor.

31. The debugging method as set forth in claim 30, wherein a process window for displaying the same source program of the processor elements or displaying different source programs of the processor element in parallel is generated on the display.

32. A debugging system for use with a data parallel processing apparatus having a plurality of processor elements for processing a program in parallel, the debugging system comprising:

a plurality of sequential debuggers to debug process programs of the processor elements, wherein each processor element has a local memory to store data peculiar to each said processor element and a common memory to store data common to all processor elements;

a management processor to output a debug command to said sequential debuggers, causing said sequential debuggers to debug the process programs, to receive reply information therefrom as the debugged results, and to manage a debugging process for the data parallel processing apparatus by adding identification to the reply information representing that the reply information from each respective sequential debugger has been received; and

a window, formed on a display of said management process, to display the same source program or different source programs of the processor elements.

33. A readable storage medium for storing a program for causing a computer to perform:

(a) outputting a debug command to a plurality of process elements for processing the same parallel program, wherein each process element has a local memory to store data peculiar to each said process element and a common memory to store data common to all process elements; and

(b) receiving reply information from a sequential debugger to debug data parallel processed programs of the process elements corresponding to the debug command that has been output at the operation (a) and to manage a debugging process of the process programs by adding identification to the reply information representing that the reply information from each respective processing element has been received.

34. A debugging system for use with a data parallel processing apparatus having a plurality of processor elements for processing the same process in parallel, the debugging system comprising:

a plurality of sequential debuggers to debug process programs of the processor elements, wherein each processor element transmits reply information in response to receipt of a debug command such that during execution of the respective debug process programs each processor adds identification information to the reply information indicating that the reply informa-

tion from each respective sequential debugger has been received; and

a management processor to output the debug command to said sequential debuggers, to cause said sequential debuggers to debug the process programs, and to receive the reply information therefrom as debugged results.

35. A debugging system for use with a data parallel processing apparatus having a plurality of processor elements for processing the same process in parallel, the debugging system comprising:

a plurality of sequential debuggers to debug process programs of the processor elements, wherein each processor element transmits reply information in response to receipt of a debug command;

a management processor to output the debug command to said sequential debuggers, to cause said sequential debuggers to debug the process programs; and

a server processor to control a corresponding sequential debugger in response to an instruction of said management process such that the server processor adds identification information to the reply information indicating that the reply information from each respective sequential debugger has been received.

* * * * *

United States Patent [19]

Censier et al.

[11] **4,012,717**

[45] **Mar. 15, 1977**

[54] **BI-PROCESSOR DATA HANDLING SYSTEM INCLUDING AUTOMATIC CONTROL OF EXCHANGES WITH EXTERNAL EQUIPMENT AND AUTOMATICALLY ACTIVATED MAINTENANCE OPERATION**

[75] Inventors: **Lucien Censier**, Conflans; **Alice Maria Recoque**, Chatenet Malabry, both of France

[73] Assignee: **Compagnie Internationale pour l'Informatique**, Louveciennes, France

[22] Filed: **Apr. 23, 1973**

[21] Appl. No.: **353,424**

[30] **Foreign Application Priority Data**

Apr. 24, 1972 France 72.14418

[52] **U.S. Cl.** .... **340/172.5**; 235/153 AE

[51] **Int. Cl.**^{2} **G06F 11/00**; G06F 15/16; G06F 15/46

[58] **Field of Search** .... 340/172.5; 235/153 AE; 444/1

[56] **References Cited**

UNITED STATES PATENTS

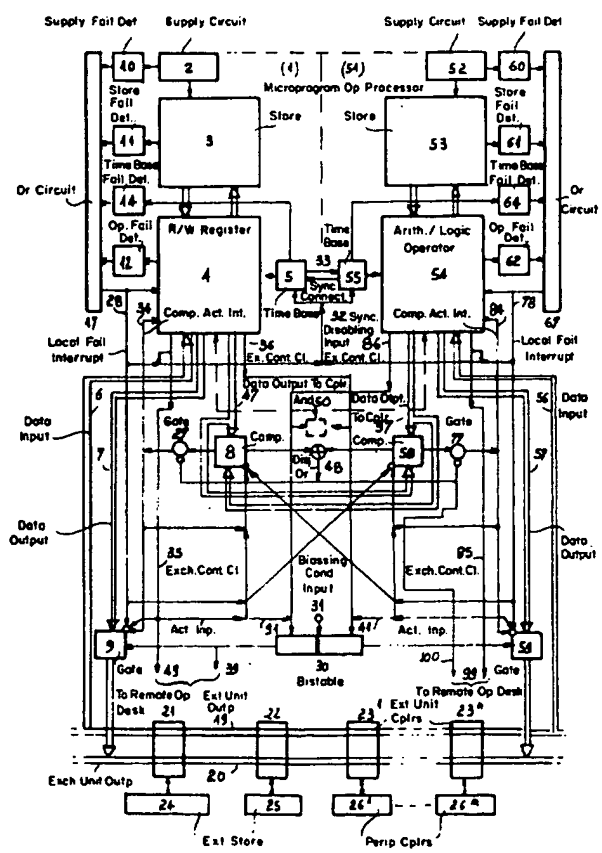
3,303,474	2/1967	Moore et al.	340/172.5
3,409,877	11/1968	Alterman et al.	340/172.5
3,444,528	5/1969	Lovell et al.	235/153 AE
3,471,686	10/1969	Connell	235/153 AE
3,517,171	6/1970	Avizienis	340/172.5
3,517,174	6/1970	Ossfeldt	235/153 AE
3,544,777	12/1970	Winkler et al.	235/153 AE
3,623,011	11/1971	Baynard, Jr. et al.	340/172.5
3,770,948	11/1973	Caputo et al.	235/153 AE

Primary Examiner—Gareth D. Shaw
Assistant Examiner—John P. Vandenburg
Attorney, Agent, or Firm—Kemon, Palmer & Estabrook

[57] **ABSTRACT**

Two microprogrammed processors of identical hardware and software facilities are synchronously operated for exchange operations with a plurality of external store and peripheral units. The exchange connections from the processors to the external units pass through gate circuits one of which only is operative at a time. Each processor has an output at which all processed words, whether instructions, microinstructions, operands or results are successively available. Comparator means permanently compare such words from the two processors and, when a mismatch occurs, block the gate circuits and their own comparison outputs and control execution in both processors of a failure detection and check up routine. Each processor includes exchange control claiming and exchange control disclaiming outputs. When a processor activates its exchange control disclaiming output, its gate means are blocked, the output of the comparator means is inhibited and the other processor is automatically qualified to proceed with the exchanges. Casually, the processors are then desynchronized.

7 Claims, 1 Drawing Figure



ands and instruction and microinstruction codes during a task of the processors. Consequently, in such an organization of the system as herein above described, a very early detection of failure will be ensured. If, further, some of the circuits such as **10** to **12** and **14** or **60** to **62** and **64** itself fails, the detection of the failure in the processor which it normally ensures, will be, just later, be detected by the comparator means proper.

As already described too, when both comparators issue a mismatch signal, a check up routine is initiated in each processor, said check up routine beginning by a task interrupt microprogram for preserving the context of the interrupted task and continuing by a systematic check up of all the functional components of the processors. Normally, at the end of such a check up routine, one of the processors must have already activated its control disclaiming output, **35** for **1** or **85** for **51**, thus controlling inhibition of the synchronization in the time bases **5** and **55** and inhibition of the comparator means (more definitely continuation of such an inhibition of the comparator means); when **35** is activated, it further inhibits the gate means **9** and applies to the input **91** of the two condition member **30** a signal requesting the switching of the condition of said member, if necessary for giving the exchange control to the other processor; when **85** is activated, it also inhibits the gate means **59** and applies to the input **41** of **30** a signal requesting the switching of said member to a condition giving the exchange control to processor **1**.

Thereafter, normally, the other processor must activates its exchange control claiming output, **36** for **1**, **86** for **51**, and the execution of the executed task is reinstated and ensured by the said processor.

Though of doubtful possibility, it may be that, once the check up routines executed, both processors still claim the control of exchanges with the peripheral and external store units. A more elaborated program of test must be requested and executed for deciding which processor is actually the subject of a failure. This may be ensured as follows:- an AND-circuit **50** is connected across the outputs **36** and **86** of the processors and it must be understood that said AND-circuit is only unblocked from the acquit instruction of a chek up routine and is normally inoperative, or is unblocked from the simultaneous activation of the inputs **34** and **84** of the comparators:- for instance, **50** may include a bistable member set to work at simultaneous activations of **34** and **84**, or of **36** and **86**, reset to rest when one of such activations disappers. The output of **50** is connected to respective test program inputs of the processors, and consequently such a program of test is requested each time the AND-circuit **50** activates its output. It may be noted that such a program of test may have recourse to an external unit since the connections **6** and **56** from the external units to the processors are never blocked and one at least of the gate means **9** and **59** is unblocked (or else, the request of such a program from an external unit may be directly provided by the activation of the output of **50**). Such a program will check the processors for execution of typical instructions and, normally again, one of the processors must have activated its control disclaiming output before the last instruction of said programme. In the utmost improbable case such a program could not obtain such a result, the task will be continued with "random" attribution of the exchange control to one of the processors as herein above explained for defining the initial condition of the two-condition member **30**.

As apparent, once the comparators inhibited, they remain inhibited until the interrupted task is completed and, for complete security in this respect, each inhibiting input of the comparators may be made a memory preserving one (a more bistable member actuated from activation of such an inhibiting input will suffice in this respect).

What is claimed is:

1. A data processing system comprising the combination of two microprogram operated processors of identical hardware and software facilities and identical coupling connections to a plurality of external store and peripheral units, gate means in each of the connections outputting from the processors to said external units, two-condition circuit means for controlling said gate means in reverse conditions of conduction, means synchronizing the operations of the processors during execution of a task, outputs in said processors at which the processed data are simultaneously available in their sequence of occurrence, comparator means having inputs connected to said processor outputs and an output responsive to a mismatch between the inputting data thereof connected to inhibiting inputs of the said gate means and to respective interrupt inputs of the processors initiating simultaneous check up routine therein and further outputs of the processors to the said two-condition circuit means individually responsive to the result of the completion of such a check up routine in each processor for controlling the said two-condition circuit means and masking the output of the said comparator means for the completion of the interrupted task after completion of the said routine.

2. A data processing system according to claim 1, wherein each processor further comprises local failure detecting circuitry and a task interrupt terminal responsive to the activation of said circuitry for disabling said synchronizing means and said comparator means and controlling the condition of the said two-condition circuit means for blocking the gate means of the processor wherein a failure identification routine is initiated.

3. A data processing system comprising a central unit, a plurality of external store and peripheral units, a common inputting channel from said external units to said central unit and a common outputting channel from said central unit to said external units, wherein:

- said central unit comprises first and second simultaneously operating, time-base synchronized microprogrammed processors of identical hardware and software organizations,
- each processor having a data output connection to the said common outputting channel through data transfer gate means,
- each processor further having a bus output on which all the codes of the words involved in the execution of any task successively appear,
- each processor further having a first task interrupt input the activation of which initiates a checkup routine in the processor, and first and second checkup result responsive outputs, one of which is activated for a positive result of the checkup and the other of which is activated for a negative result of the checkup,
- a two-condition member having distinct actuation inputs and a pair of complementary condition outputs respectively controlling in reciprocal transfer conditions the data transfer gate means of said first and second processors, one of the said actuation

inputs being connected to the first checkup responsive output of the first processor and to the second checkup responsive output of the second processor and the other one of the said actuation inputs being connected to the second checkup responsive output of the first processor and to the first checkup responsive output of the second processor, and,

code comparator means having respective inputs connected to the bus outputs of the first and second processors and a mismatch output connected to both the said first task interrupt inputs of the processors, to inhibiting inputs of both the said data transfer gates and to a code comparator means self-inhibiting input.

4. A data processing system according to claim 3, wherein said code comparator means comprise first and second code comparator circuits each having inputs connected to the bus outputs of the processors, each having its output connected to its own self-inhibiting input through a gate connection, the first code comparator circuit having said gated output connected to the first task interrupt input of the first processor and to the inhibiting input of the data transfer gate means of said first processor, the second code comparator circuit having its gated output connected to the first task interrupt input of the second processor and to the inhibiting input of the data transfer gate of said second processor, and wherein said first and second code comparator circuits have outputs connected to respective inputs of an exclusive-OR circuit the output of which is connected to inhibit inputs of the said gated outputs on the occurrence of a mismatch condition between the inputs of the said exclusive-OR circuit.

5. A data processing system according to claim 3, wherein each of the said first and second processors includes a local failure detecting orgaination, the output of which is connected to a second task interrupt input the activation of which initiates a failure localization routine in the processor, said second task interrupt input being connected to an inhibit input of the data transfer means of said processor, to the inhibit input of the said code comparator means and to the actuation input of the said two-condition member which is connected to the second checkup responsive output of the processor.

6. A data processing system according to claim 5, wherein both second checkup responsive outputs and both second task interrupt inputs of the processors are further connected to a time-base synchronizing inhibiting input of the time bases of the processors.

7. A data processing system according to claim 3, wherein a diagnostic routine control circuit is connected across the said first checkup responsive outputs of the processors and has its output connected to diagnostic routine initiating inputs of the said processors.

* * * * *

```
Set    Items   Description
S1     5250336   RUN OR RUNS OR RUNNING? OR EXECUT? OR PROCESS? OR BEGIN? OR
                 START? OR COMMENC?
S2     1461910   APPLICATION? OR SOFTWARE? OR PROGRAM? OR ROUTINE? OR SUBRO-
               UTINE?
S3       44677   APP? ? OR SUBPROGRAM? OR MACRO? ? OR UTILIT?
S4     2545489   "SAME" OR SIMILAR? OR IDENTICAL? OR ALIKE? OR EQUAL? OR EQ-
               UIVALEN? OR CLONE?
S5     6774253   TWO OR DIFFERENT? OR SECOND? OR 2ND OR ANOTHER OR PARALLEL?
                 OR DUAL? OR TWIN? OR DOUBL? OR PAIR? ?
S6     1362744   COMPUTER? OR SERVER? OR WORKSTATION? OR PROCESSOR? OR PROC-
               ESSING()(SYSTEM? OR DEVICE?)
S7      255352   CPU? ? OR CENTRALPROCESSOR? OR DATAPROCESSOR? OR MICROPROC-
               ESSOR?
S8     1507248   DIAGNOS? OR AUDIT? OR TRACK? OR RESPON? OR ADVIS? OR ADVIC?
                 OR SUGGEST? OR PRESCRIB?
S9        2424   TROUBLESHOOT? OR (FIND? OR LOCAT? OR DETERMIN? OR RESPON?)-
               (2N)(PROBLEM? OR SLOWDOWN? OR BUG? ?)
S10    3601485   PROBLEM? OR ANOMAL? OR ABNORMAL? OR DISCREPANC? OR GAP? ?
S11     252096   PRIORIT? OR RANK? OR SORT? OR CLASSIF? OR CATEGOR? OR HIER-
               ARCH?
S12    1981489   REMEDY? OR REMEDIE? OR CURE? OR FIX OR DEBUG? OR FIXES OR -
               FIXED OR CURATIV? OR CURING?
S13    3243491   AMELIORAT? OR IMPROV? OR HEAL? OR ACCELERAT? OR SPEED?()UP
               OR BETTER? OR PRESCRIPTION? OR QUICKEN?
S14    1136820   SUBTRACT? OR MINUS? OR DECREMENT? OR LESSEN? OR LESS OR MU-
               LTIPLY? OR MULTIPLICAT?
S15      44060   CLOCK()(TICK? OR TIME? OR RATE?) OR PROCESS?()(TIME? OR LE-
               NGTH?) OR CLOCKTIME? OR CLOCKRATE? OR CLOCKTICK?
S16    1254296   IC=G06F?
S17     947854   MC=T01?
S18        598   S1(7N)S4(7N)S2:S3 AND S5(7N)S6:S7
S19          1   S18 AND S14 AND S15
S20         32   S18 AND S8:S11 AND S12:S13
S21         36   S18 AND S8:S10 AND S11:S13
S22        560   S18 AND S1:S4(7N)S5:S7 AND (S16:S17 OR S8:S15)
S23        521   S22 AND S16:S17
S24        238   S23 AND S8:S13
S25         35   S24 AND S11
S26         37   (S18 OR S22:S23) AND S11
S27         62   S19:S21 OR S25:S26
S28     829010   PR=2002:2005
S29         16   S22 AND S2:S3/TI AND S8:S13/TI
S30         71   S27 OR S29
S31         69   S30 NOT S28
S32         69   IDPAT (sorted in duplicate/non-duplicate order)
File 347:JAPIO Nov 1976-2005/Jun(Updated 051004)
         (c) 2005 JPO & JAPIO
File 350:Derwent WPIX 1963-2005/UD,UM &UP=200570
         (c) 2005  Thomson Derwent
```

PAT LIT

BIBLIOG

FILES

32/3,K/30 (Item 30 from file: 350)
DIALOG(R)File 350:Derwent WPIX

007261151
WPI Acc No: 1987-258158/198737
XRPX Acc No: N87-193309
Multi- processor **central controller for communications exchange - uses** parallel **central** processors **each incorporating fault recognition circuit and specific fault** diagnosis programme
Patent Assignee: SIEMENS AG (SIEI)
Inventor: BITZINGER R; ENGL W; HUMML S; SCHREIER K; HUMMI S; SCHFREIER K
Number of Countries: 017 Number of Patents: 011
Patent Family:

Patent No	Kind	Date	Applicat No	Kind	Date	Week	
EP 236803	A	19870916	EP 87102240	A	19870217	198737	B
FI 8701058	A	19870913				198748	
DK 8701246	A	19870913				198803	
BR 8701106	A	19871229				198806	
PT 84444	A	19880303				198814	
CN 8701838	A	19871202				198848	
US 4914572	A	19900403	US 8724759	A	19870311	199019	
EP 236803	B	19920115				199203	
DE 3775946	G	19920227				199210	
FI 89442	B	19930615	FI 871058	A	19870311	199328	
DK 167333	B	19931011	DK 871246	A	19870311	199346	

Priority Applications (No Type Date): DE 3625498 A 19860728; DE 3608261 A 19860312
Patent Details:

Patent No	Kind	Lan	Pg	Main IPC	Filing Notes
EP 236803	A	G	8		
Designated States (Regional): AT BE CH DE FR GB GR					
EP 236803	B				
Designated States (Regional): AT BE CH DE FR GB GR IT LI NL SE					
FI 89442	B			H04Q-003/545	patent FI 8701058
DK 167333	B			H04Q-003/545	patent DK 8701246

Multi- processor **central controller for communications exchange...**

...**uses** parallel **central** processors **each incorporating fault recognition circuit and specific fault** diagnosis programme

...Abstract (Basic): The controller has a number of central **processor** (CP,10C) and a central memory (CMY) coupled in parallel to a central bus system (B:CMYO;B:CMY1). The **processors** (CP,10C) employ **parallel processing** units (PU) and a fault recognition circuit (V) using a function checking programme held in...

...Each respective **processor** (CP) is uncoupled from the bus system (B:CMY0;B:CMY1) upon detection of an operating fault with initiation of a fault **diagnosis** programme via the processing units (PU...

...USE - For fault-tolerant multi- **processor** system...
...Abstract (Equivalent): a switching system, preferably of a telephone switching system, in which a plurality of central **processors** (CP,IOC), which each contain **dual** **processor** units (PU) which are microsynchronously operated in **parallel** -apart from a certain slip which may be tolerated - and carry out the actual fault...

...comparator circuit (V), for the immediate checking of the instructions and/or data of both **processor** units (PU) of the respective **processor** being **processed** in each case, and which have their own local memory (LMY,LMY:IO) with a...

...say PROM part for example, this ROM part in turn storing, if one compares the **different** **processors** (CP,OIC), in each case at least partially **similar** part **programs** for the self testing of the respective **processor** (CP,IOC),as well as further part switching programs, for example the switching **program** sections required by this **processor** (CP,IOC) in each case the most frequently and/or the most quicklu, as well as a central memory (CMY), to which the central **processors** (CP,IOC) have access via the bus system (B:CMY,OB:CMY1), and which store...

...that are required more seldomly and/or in each case not immediately by a central **processor** (CP,IOC) as well as - at least temporarily stored - data on a large number of...

...connections and on peripheral system elements which are accessible to several or to all the **processors** (CP,IOC), are connected in **parallel** to a central bus system (B:CMYO/B:CMY1), characterised in that, after an error detected by at least one of the error detection circuits (V) of a **processor** (for example CPx) in the respective **processor** (CPx), at least if it cannot by immediately corrected, first of all the
...Abstract (Equivalent): The method of error protection has number of central **processors** (CP,IOC) as well as a central memory (CMY) connected in parallel to a central bus system (B:CMYO/B:CMY1). The **processors** includes **dual** highly-synchronous **parallel** driven **processor** units (PU) and integral error detection circuits (V), as well as an integral memory (LMY), in the ROM-area of which test **program** sections are stored for testing the respective **processors** (CP,IOC...

...of an error by at least one of the error detection circuits (V) of a **processors** , in respective **processor** (CPx), at least if the error is not immediately correctable, the error detection circuit (V in CPx) **starts** isolating the respective **processor** (CPx) from the bus system (B:CMY). The respective **processor** (CPx) **starts** the read-out of the test program sections, stored in its own local memory (LMY...
...Title Terms: **PROCESSOR** ;
International Patent Class (Additional): **G06F-007/00** ...

... **G06F-011/16** ...

... **G06F-015/16**
Manual Codes (EPI/S-X): **T01-G03** ...

... **T01-J02C**

United States Patent [19]

Bitzinger et al.

[11] **Patent Number: 4,914,572**

[45] **Date of Patent: Apr. 3, 1990**

[54] **METHOD FOR OPERATING AN ERROR PROTECTED MULTIPROCESSOR CENTRAL CONTROL UNIT IN A SWITCHING SYSTEM**

[75] Inventors: **Rudolf Bitzinger**, Munich; **Walter Engl**, Feldkirchen; **Siegfried Humml**, Penzberg; **Klaus Schreier**, Penzberg, all of Fed. Rep. of Germany

[73] Assignees: **Siemens Aktiengesellschaft**, Berlin and Munich, Fed. Rep. of Germany

[21] Appl. No.: **24,759**

[22] Filed: **Mar. 11, 1987**

[30] **Foreign Application Priority Data**

Mar. 12, 1986 [DE] Fed. Rep. of Germany 3608261
Jul. 28, 1986 [DE] Fed. Rep. of Germany 3625498

[51] **Int. Cl.**[4] **G06F 11/00**; G06F 15/16

[52] **U.S. Cl.** **364/200**; 364/228.1; 364/228.3; 364/265; 364/267; 364/267.4; 364/267.7; 364/268.3; 364/244.6; 379/10; 379/269; 379/279

[58] **Field of Search** ... 364/200 MS File, 900 MS File; 379/10, 269, 279

[56] **References Cited**

U.S. PATENT DOCUMENTS

3,576,541	4/1971	Kwan	364/200
3,768,074	10/1973	Sharp et al.	364/200
3,814,919	6/1974	Repton et al.	364/200
3,864,670	2/1975	Inoue et al.	364/200
3,898,621	8/1975	Zelinski et al.	364/200
4,371,754	2/1983	De et al.	371/10
4,390,953	6/1983	Johnstone	364/900
4,439,826	3/1984	Lawrence	364/200
4,443,849	4/1984	Ohwada	364/200
4,453,210	6/1984	Suzuki et al.	364/200
4,751,703	6/1988	Picon et al.	371/10

FOREIGN PATENT DOCUMENTS

33347921	11/1984	Fed. Rep. of Germany .
2106176	9/1979	United Kingdom .

OTHER PUBLICATIONS

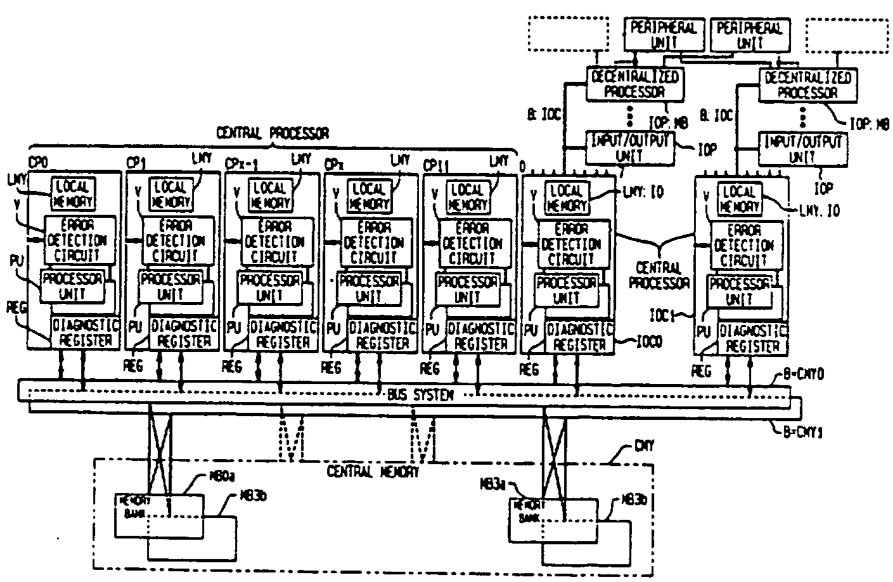
IEEE Intl. Conf. on Comm., May, 1984, Beuscher et al.
Telecom Report, Mar./Apr. 1984, No. 2, Knapek et al.
GTE Journal, vol. 20, Mar./Apr. 1982, Bassett et al.
8th Annual Symposium on Comp. Architecture, May 1981, Geitz et al.

Primary Examiner—Raulfe B. Zache
Assistant Examiner—Emily Y. Chan
Attorney, Agent, or Firm—John F. Moran

[57] **ABSTRACT**

A method provides error protection in a multiprocessor central control unit of a switching system wherein a number of central processors (CP, IOC) as well as a central memory (CMY) are connected in parallel to a central bus system (B:CMY0/B:CMY1). The processors include dual highly-synchronous parallel driven processor units (PU) —apart from a possible tolerable positive timing slip—and integral error detection circuits (V), as well as an integral local memory (LMY), in the ROM-area of which test program sections are stored for testing the respective processors (CP, IOC). Upon the detection of an error by at least one of the error detection circuits (V) of a processor (for example CPx), in the respective processor (CPx), at least if the error is not immediately correctable, the error detection circuit (V in CPx) starts isolating the respective processor (CPx) from the bus system (B:CMY). The respective processor (CPx) starts the read-out of the test program sections, stored in its own local memory (LMY), for localizing and identifying the error source and/or the defect causing such errors.

10 Claims, 1 Drawing Sheet



ples and general characteristics are previously known and described in detail in these prior patent applications, it is unnecessary to describe the construction and operation of the present central control unit in complete detail again. Instead it is adequate to discuss only upon the particular matters and techniques as they relate in accordance with the present invention.

DETAILED DESCRIPTION

The sole FIGURE includes central processors designated CP0 . . . CP11, IOC0, IOC1 . . . , with integrated redundant processor building blocks, PU, e.g. with integrated double 32-bit microprocessor chips PU. These processor units PU are driven in synchronous step in parallel, apart from a possible tolerable timing slip, and therefore carry out the actual error protected interconnects, for which purpose they contain, or have been associated with, at least one independent error detection circuit, e.g. EDC circuits, parity-bit networks and/or comparison circuits V, particularly for the immediate checking of instructions which includes commands and/or data being processed by both processor units PU, of the respective processor. Each of the processors CP, IOC contains individual local memory LMY, LMY:IO, with a RAM section and in particular a ROM section, i.e. a PROM section, which store at least partially similar error diagnostic programs namely test program sections for self testing the respective processor, and which store switching program sections, in particular the most frequent and/or most quickly needed switching program sections required by the respective processor CP, IOC.

The central memory CMY, to which the central processors CP, IOC have access over the dual bus system B:CMY0/B:CMY1, store at least various seldom and/or not immediately needed switching program sections required by a central processor CP, IOC, as well as data—accessible at least temporarily—for several or for all processors CP, IOC which data concern a multiplicity of connections existing at that time, and concern peripheral system device features.

As soon as an error is detected by at least one of the error detection circuits, for instance, by one of the processor comparison circuits V in the respective processor, for example in CPx, at least if this error is not easily correctable, the output signal of the respective error detection circuit, starts a process to isolate the respective processor CPx from the bus system B:CMY0/B:-CMY1, e.g. by means of an I/O unit, and starts the read-out of the respective test program section for such trouble, stored in the ROM-part of the local memory LMY (and/or LMY:IO). Thereupon the two processor units PU of this processor CPx begin to process this test program for the more specific location and/or typfication of the respective error or of the defect which causes such errors. Through the isolation of the respective processor CPx, the propagation of the error through the entire switching system is avoided. In other words, a high tolerance for errors is achieved. The immediately introduced self testing simplifies the subsequent error diagnosis for maintenance service staff, and the possible repair necessary is very specific, and in general limited to a small portion or section of the respective processor. In the event that no indication of an error/defect is established during test processing, the respective processor CPx reconnects itself again, preferably on its own, with the bus system B:CMY0/B:-CMY1, so that the original extremely high operational availability is restored to provide a minimum of downtime.

The subsequent failure diagnosis of the defective processor is also especially rapidly and simply possible if, in the event that an indication of an error/defect is established during test processing, an error code corresponding to this error/defect for example, the address of the related test program section command, is stored in a diagnostic register REG of the respective processor CPx.

When an error/defect has been adequately localized and/or typified, it is even possible, immediately after storing the error code, to interrupt processing of the test program.

If the defective processor CPx is simply left isolated from the bus system B:CMY it may immediately be by-passed during the normal distribution of switching commands so that the isolation does not result in a loss of time for the entire switching system whereby the effect of the isolation on the operation of the central control unit remains very limited.

The remote diagnosis of all errors/defects and the corresponding appropriate preparation for the subsequent servicing of the central control unit is possible, by means of a special processor, at an operation and maintenance station for example, which may access over the bus system B:CMY0/B:CMY1 at least to each respective processor CPx, although this respective processor CPx is isolated from the bus system B:CMY0/B:CMY1, for interrogating the contents of its diagnostic register REG via a special interrogating code.

In the event that a direct indication of the error/-defect of only one of the two processor units PU occurred, e.g. by an associated EDC circuit or an associated parity bit network of this respective processor unit PU, this respective processor unit PU alone, may be isolated and normal processor operation may be maintained via the other processing unit PU of this respective processor, thus leaving the central control unit in its previous high state of availability. It is also desirable, during that isolation of the single processor unit PU, to log an indication of this condition for the maintenance service staff, for example, again in the diagnostic register REG which may even be remotely interrogated by the operation and maintenance station. The repair of this processor may then be included in preparing procedures for the subsequent maintenance.

There has thus been shown and described a novel arrangement and technique for providing error protection in multiprocessor central control unit of a switching system which fulfills all the objects and advantages sought therefore. Many changes, modifications, variations and other uses and applications of the subject invention will, however, become apparent to those skilled in the art after considering this specification and the accompanying drawing which disclose the preferred embodiments thereof. All such changes, modifications, variations and other uses and applications which do not depart from the spirit and scope of the invention are deemed to be covered by the invention which is limited only by the claims which follow.

We claim:

1. A method of operating an error protected high availability multiprocessor serving as a central controller of a switching system providing inter-connections to subscribers, particularly a telephone switching system, the central controller comprising:

(a) a plurality of central processors (CP, IOC), each central processor including dual, apart from a possible tolerable timing slip, parallel synchronously driven processor units (PU) for carrying out the inter-connections to subscribers connected to said switching system, including at least one integral error detection circuit (V) for immediately checking instructions processed by both of the dual processor units (PU) of the respective processor, and including a local memory (LMY, LMY:10) having a ROM-section, storing test program sections for self testing of the respective processor (CP, IOC), and storing switching program sections required most frequently and most quickly by the respective processor (CP, IOC),

(b) a central main memory (CMY) including a ROM-area storing at least seldom and not immediately required (CP, IOC) switching program sections, and including a memory-area storing at least temporarily data accessible for a number of or for all processors (CP, IOC), whereby such data concern inter-connections between subscribers and concern peripheral system elements, and

(c) a central bus system (B:CMY) to which are connected in parallel the processors (CP, IOC) and the main memory (CMY), after detecting an error by at least one of the error detection circuits (V) or a processor (CPx, for example), at least if this error is not immediately correctable, the method comprising the steps:

(a) isolating the respective processor (CPx) from the bus system (B:CMY);

(b) starting to read-out test program sections stored in ROM-section of its own local memory (LMY, LMY:IO) by the respective processor (CPx); and

(c) processing this test program for localization and identification of the error and a defect causing such errors by the respective processor (CPx).

2. A method in accordance with claim 1, further comprising the step:

(d) reconnecting the respective processor (CPx) to the bus system (B:CMY) if no indication of an error and/or defect is detected during the processing of the test program.

3. A method in accordance with claim 1, further comprising the step:

(d) storing an error code obtained from its own local memory corresponding to this error/defect in an integral diagnostic register (REG) of the respective processor (CPx) if an indication of an error and/or defect is detected during processing of the test program.

4. A method in accordance with claim 3, said error code indicative of an address of that test program command or of those test program commands, at which the respective error defect was localized and identified.

5. A method in accordance with claim 3, further comprising the step:

(e) interrupting the processing of the test program by the respective processor (CPx) after the error code is stored.

6. A method in accordance with claim 3, further comprising the step:

(f) maintaining isolation of the respective processor (CPx) after the error code is stored.

7. A method in accordance with claim 6, further comprising accessing each processor (CPx) isolated from the bus system (B:CMY) via a special code by a particular processor and interrogating the contents of the diagnostic register (REG) of this isolated processor (CPx) to diagnose the processor being isolated.

8. A method in accordance with claim 1, further comprising the step:

directly monitoring each of the dual processor units (PU) by each error detection circuit and directly indicating an error or a defect in a respective one of the dual processor units (PU isolating the respective processor unit (PU) alone, whereby the normal processor operations are maintained via the other processor unit (PU) along within this respective processor.

9. A method in accordance with claim 8, further comprising storing an indication of this isolated one of the processor units (PU) in a diagnostic register of the respective processor.

10. A method in accordance with claim 5, wherein a special processor accesses each processor (CPx) isolated from the bus system (B:CMY0/B:CMY1) via a special code, and interrogates the contents of the diagnostic register (REG) to diagnose the processor being isolated.

* * * * *

32/3,K/36 (Item 36 from file: 350)
DIALOG(R)File 350:Derwent WPIX

004099039
WPI Acc No: 1984-244580/198440
XRPX Acc No: N84-182947

Safe operation of process controller - feeding command data to processors **to** execute **test** routines **before being allowed to be transmitted as output**

Patent Assignee: SIEMENS AG (SIEI)
Inventor: HOMEISTER M; RAIMER J
Number of Countries: 005 Number of Patents: 005
Patent Family:

Patent No	Kind	Date	Applicat No	Kind	Date	Week	
DE 3310975	A	19840927	DE 3310975	A	19830325	198440	B
EP 120339	A	19841003	EP 84102198	A	19840301	198440	
DE 3462231	G	19870305				198710	
EP 120339	B	19870128				199127	
EP 120339	B2	19910703				199127	

Priority Applications (No Type Date): DE 3310975 A 19830325
Patent Details:

Patent No	Kind	Lan	Pg	Main IPC	Filing Notes
DE 3310975	A		15		
EP 120339	A	G			

Designated States (Regional): AT CH DE LI NL

Patent No	Kind	Lan	Pg	Main IPC	Filing Notes
EP 120339	B	G			

Designated States (Regional): AT CH DE LI NL

Patent No	Kind	Lan	Pg	Main IPC	Filing Notes
EP 120339	B2				

Designated States (Regional): CH DE LI NL

... **feeding command data to** processors **to** execute **test** routines **before being allowed to be transmitted as output**

...Abstract (Basic): The data **processing** module has **two** independently operating microcomputers (MC1, MC2) and a safety relay logic unit (RV) when a command...

...in a memory in the output interface (AE) and is transmitted to both microcomputers that **execute** a check **routine** . Provided that the **same** result is reached the command is allowed to be transmitted to output. When an auxiliary command mode is employed the **second processor** (MC2) outputs to a special display (KA) for operator verification. Only when an input key...

...A device for the reliable **process** control employing **two** microcomputers which are independent of one another and do not operate so as to be...

...other microcomputer (MC2) by means of a safety-oriented input doubler (EV), that both microcomputers **classify** the data, which are fed thereto, independently of **another** in accordance with the respectively present **process** control instruction and feeds the **classification** results to a relay connection (RV) which when the **classification** ...

...results of the **two** microcomputers are **identical** causes the...

...release of the data stored in the output device (AE), if the

respectively **classified** process control instruction relates to a control operation, but during common recognition of an auxiliary

...Abstract (Equivalent): Device for reliable **process** control employing **two** microcomputers (MC1,MC2) which are independent of another, do not operate in a safety-oriented...

...microcomputer (MC2) via a safety-oriented input doubler (EV), in that both microcomputers (MC1,MC2) **classify** independently of one another the data fed to them in accordance with the type of the respective process control instruction present and feed the **classification** results to a relay connection (RV) which, when the **classification** results of the two microcomputers match, initiates the release of the data stored in the output device (AE) if the respective **classified** process control instruction relates to a control operation, whereas in the case of joint recognition...

...an auxiliary operation is concerned, converts the command data sent to it vai the input **doubler** (EV) into the corresp. **process** control instru

...Title Terms: **PROCESSOR** ;

Manual Codes (EPI/S-X): **T01-J07** ...

DERWENT-ACC-NO: 1984-244580

DERWENT-WEEK: 199736

TITLE: Safe operation of process controller - feeding command data to processors to execute test routines before being allowed to be transmitted as output

INVENTOR: HOMEISTER, M; RAIMER, J

PATENT-ASSIGNEE: SIEMENS AG[SIEI]

PRIORITY-DATA: 1983DE-3310975 (March 25, 1983)

PATENT-FAMILY:

PUB-NO	PAGES	MAIN-IPC	PUB-DATE	LANGUAGE
DE **3310975** A	015	N/A	September 27, 1984	N/A
EP 120339 A	000	N/A	October 3, 1984	G
DE 3462231 G	000	N/A	March 5, 1987	N/A
EP 120339 B	000	N/A	January 28, 1987	G
EP 120339 B2	000	N/A	July 3, 1991	N/A

DESIGNATED-STATES: AT CH DE LI NL AT CH DE LI NL CH DE LI NL

CITED-DOCUMENTS: CH 535154; DE 2303828 ; 5.Jnl.Ref

APPLICATION-DATA:

PUB-NO	APPL-DESCRIPTOR	APPL-NO	APPL-DATE
DE 3310975A	N/A	1983DE-3310975	March 25, 1983
EP 120339A	N/A	1984EP-0102198	March 1, 1984

INT-CL (IPC): B61L027/00, G05B009/03 , G05B023/02

ABSTRACTED-PUB-NO: DE 3310975A

BASIC-ABSTRACT:

The process control system has a data input terminal (OE) that allow communication with a process via a data processing module (OV) and an output interface (AE). Within the input terminal is an input keyboard (ET), and a V.U (SG).

The data processing module has two independently operating microcomputers (MC1, MC2) and a safety relay logic unit (RV) when a command is generated the data are held in a memory in the output interface (AE) and is transmitted to both microcomputers that execute a check routine. Provided that the same result is reached the command is allowed to be transmitted to output. When an auxiliary command mode is employed the second processor (MC2) outputs to a special display (KA) for operator verification. Only when an input key (FS) is actuated is the command executed.

ADVANTAGE - Allows safe operation to be imposed on auxiliary mode.

ABSTRACTED-PUB-NO: EP 120339A

EQUIVALENT-ABSTRACTS:

A device for the reliable process control employing two microcomputers which are independent of one another and do not operate so as to be safety-oriented and which commonly act upon the process which is to be controlled, and allow both control operations, whose reliability is tested in a separate safety plane outside the microcomputer, and also auxiliary operations, whose reliability is no longer tested, to be carried out, in particular for the control of a railroad signalling device of at least one operating location, characterised in that the one microcomputer (MC1) converts the process control instructions which are present in order to be carried out, into corresponding command data,

and stores them in an output device (AE) and re-reads the stored data, where
the re-read data are simultaneously fed to the other microcomputer (MC2) by
means of a safety-oriented input doubler (EV), that both microcomputers
classify the data, which are fed thereto, independently of another in
accordance with the respectively present process control instruction and feeds
the classification results to a relay connection (RV) which when the classification

results of the two microcomputers are identical causes the

release of the data stored in the output device (AE), if the respectively
classified process control instruction relates to a control operation, but
during common recognition of an auxiliary operation makes the release of the
data stored in the output device dependent upon a separate agreement of an
operator, which is fed via the relay connection (RV).

(8pp)

EP 120339B

Device for reliable process control employing two microcomputers (MC1,MC2)
which are independent of another, do not operate in a safety-oriented manner
and act jointly on the process to be controlled, and in doing so permit the
execution of both control operations whose admissibility is checked on a
separate safety level outside the microcomputer, and auxiliary operations whose
admissibility is not checked further, and of which one (MC1) converts the
process control instructions present for execution in each case into corresp.
command data and forwards them to an output device (AE), via which they can be
switched further to the process, the command data being temp. stored and
forwarded to an operator for checking in the case of an auxiliary operation, in
partic., for controlling a railway signalling device from at elast one

operating terminal, characterised in that one of the microcomputers (MC1)
re-reads the data stored in the output device (AE), the re-read data being
simultaneously also fed to the other microcomputer (MC2) via a safety-oriented
input doubler (EV), in that both microcomputers (MC1,MC2) classify
independently of one another the data fed to them in accordance with the type
of the respective process control instruction present and feed the
classification results to a relay connection (RV) which, when the
classification results of the two microcomputers match, initiates the release
of the data stored in the output device (AE) if the respective classified
process control instruction relates to a control operation, whereas in the case
of joint recognition of an auxiliary operation it makes the release of the data
stored in the output device dependent upon the agreement of the operator
notified to it, and in that the microcomputer (MC2) not directly receiving the
process control instructions in each case, where an auxiliary operation is
concerned, converts the command data sent to it vai the input doubler (EV) into
the corresp. process control instru

CHOSEN-DRAWING: Dwg.1/1

DERWENT-CLASS: Q21 T01 T06

EPI-CODES: T01-J07; T06-A03; T06-A08;

PUB-NO: EP000120339A1

DOCUMENT-IDENTIFIER: EP **120339** A1

TITLE: Device for reliable process control.

PUBN-DATE: October 3, 1984

INVENTOR-INFORMATION:

NAME	COUNTRY
HOMEISTER, MANFRED DIPL-ING	N/A
RAIMER, JURGEN ING GRAD	N/A

ASSIGNEE-INFORMATION:

NAME	COUNTRY
SIEMENS AG	DE

APPL-NO: EP84102198

APPL-DATE: March 1, 1984

PRIORITY-DATA: DE03310975A (March 25, 1983)

INT-CL (IPC): B61L027/00

EUR-CL (EPC): B61L027/00 ; B61L021/04

US-CL-CURRENT: 246/5

ABSTRACT:

1. A device for the reliable process control employing two microcomputers which are independent of one another and do not operate so as to be safety-oriented and which commonly act upon the process which is to be controlled, and allow both control operations, whose reliability is tested in a separate safety plane outside the microcomputer, and also auxiliary operations, whose reliability is no longer tested, to be carried out, in particular for the control of a railroad signalling device of at least one operating location,

characterized in that the one microcomputer (MC1) converts the process control instructions which are present in order to be carried out, into corresponding command data, and stores them in an output device (AE) and re-reads the stored data, where the reread data are simultaneously fed to the other microcomputer (MC2) by means of a safety-oriented input double (EV), that both microcomputers classify the data, which are fed thereto, independently of another in accordance with the respectively present process control instruction and feeds the classification results to a relay connection (RV) which when the classification results of the two microcomputers are identical causes the release of the data stored in the output device (AE), if the respectively classified process control instruction relates to a control operation, but during common recognition of an auxiliary operation makes the release of the data stored in the output device dependent upon a separate agreement of an operator, which is fed via the relay connection (RV).

32/3,K/28 (Item 28 from file: 350)
DIALOG(R)File 350:Derwent WPIX

008441546 **Image available**
WPI Acc No: 1990-328546/199044
XRPX Acc No: N90-251518

Programmable control unit with two processors **and program memory - has controller connected by bus with dedicated control line between processors**

Patent Assignee: YOKOGAWA ELECTRIC CORP (YOKG)
Inventor: HASEGAWA K; KADOWAKI T; KAWATA Y; MATSUOKA K; YONEZAWA M
Number of Countries: 007 Number of Patents: 011
Patent Family:

Patent No	Kind	Date	Applicat No	Kind	Date	Week	
DE 4011278	A	19901025	DE 4011278	A	19900406	199044	B
NL 9000984	A	19901116				199049	
FR 2646254	A	19901026				199050	
GB 2232514	A	19901212	GB 906947	A	19900328	199050	
CN 1046987	A	19901114				199130	
GB 2244828	A	19911211	GB 9012247	A	19900328	199150	
GB 2232514	B	19930901	GB 906947	A	19900328	199335	
GB 2244828	B	19930901	GB 9112247	A	19900328	199335	
KR 9402324	B1	19940323	KR 905768	A	19900424	199601	
KR 9402339	B1	19940323	KR 905768	A	19900424	199601	
			KR 9322821	A	19931030		
US 5553297	A	19960903	US 90513454	A	19900423	199641	
			US 93119322	A	19930909		

Priority Applications (No Type Date): JP 89275513 A 19891023; JP 89104052 A 19890424; JP 89129707 A 19890523; JP 89275512 A 19891023

Patent Details:

Patent No	Kind	Lan	Pg	Main IPC	Filing Notes
GB 2232514	B		14	G06F-009/44	
GB 2244828	B		50	G06F-009/44	
KR 9402339	B1			G06F-009/44	Div ex application KR 905768
US 5553297	A		41	G06F-015/16	Cont of application US 90513454
KR 9402324	B1			G06F-009/44	

Programmable control unit with two processors **and program memory**...

...Abstract (Basic): The programmable control unit contains an information bus connecting **two processors** and a program memory contg. interpretation type and basic or standard instructions. Signals are transferred...

...USE/ADVANTAGE - Has **improved** internal geometry and processing efficiency. (43pp Dwg.No.2/36)

...Abstract (Equivalent): commands to be executed by said processor; a data memory for temporarily storing data; a **fixed** memory for storing a self- **diagnostic** program; a communication interface participating in a communicating operation with a host computer; an I...

...internal bus for mutually connecting said processor, said 1-bit processor, said data memory, said **fixed** memory, said communication interface and said I/O interface...

...A programmable controller comprising: a first **processor** ; a **second processor** ; a program memory system for storing interpreter type program commands and basic commands; a first...

...bus through which said first processor is connected to said program memory system; and a **second** information bus through which said **second processor** is connected to said first **processor** ; whereby said first processor reads a program command and judges whether said program command be executed by said first **processor** or said **second processor** ; when said first **processor** judges that said program command be executed by said first processor, said first processor itself operates, said first processor reads said **program** command from said **program** memory preparatory to **execution** of said **program** command and at the **same** time issues a pseudo command to said **second processor** ; and when said first **processor** judges that said **program** command be executed by said **second processor** , said first **processor** supplies said program command to said **second processor** in place of said pseudo command...

...Abstract (Equivalent): a read only memory (ROM) for storing a self-**diagnostic** program...

US005553297A

United States Patent [19]

Yonezawa et al.

[11] **Patent Number: 5,553,297**

[45] **Date of Patent: Sep. 3, 1996**

[54] **INDUSTRIAL CONTROL APPARATUS**

[75] Inventors: **Masaaki Yonezawa; Kiyoshi Hasegawa; Yasunori Kawata; Kouji Matsuoka; Takasi Kadowaki**, all of Tokyo, Japan

[73] Assignee: **Yokogawa Electric Corporation**, Musashino, Japan

[21] Appl. No.: **119,322**

[22] Filed: **Sep. 9, 1993**

Related U.S. Application Data

[63] Continuation of Ser. No. 513,454, Apr. 23, 1990, abandoned.

[30] **Foreign Application Priority Data**

Apr. 24, 1989	[JP]	Japan	1-104052
May 23, 1989	[JP]	Japan	1-129707
Oct. 23, 1989	[JP]	Japan	1-275512
Oct. 23, 1989	[JP]	Japan	1-275513

[51] **Int. Cl.**^{6} **G06F 15/16**

[52] **U.S. Cl.** **395/800**; 364/132; 364/136; 364/DIG. 1

[58] **Field of Search** 395/800; 364/131–136

[56] **References Cited**

U.S. PATENT DOCUMENTS

4,303,990	12/1981	Seipp	395/740
4,592,010	5/1986	Wollscheid	395/800
4,600,988	7/1986	Tendulkar et al.	395/550
4,648,068	3/1987	Ninnemann et al.	395/281
4,750,110	6/1988	Mothersole et al.	395/375
4,797,811	1/1989	Kiyokawa et al.	364/474.23
4,858,101	8/1989	Stewart et al.	364/131
4,942,552	7/1990	Merrill et al.	395/826
4,972,365	11/1990	Dodds et al.	395/825
4,985,831	1/1991	Dulong et al.	395/650
5,068,778	11/1991	Kosem et al.	364/138
5,068,821	11/1991	Sexton et al.	395/800
5,287,548	2/1994	Flood et al.	395/375

OTHER PUBLICATIONS

MC68881/MC68882 Floating–Point Coprocessor User's Manual, 1987, pp. 7–1 to 7–38.

M68000 Family Reference, 1988, pp. –62 to 3–63, 3–66 to 3–67, 4–32 to 4–35, 4–50 to 4–51, 4–6 to 4–6, and 4–74 to 4–75.

Primary Examiner—William M. Treat
Attorney, Agent, or Firm—Pennie & Edmonds

[57] **ABSTRACT**

A programmable controller is disclosed that incorporates a sequence control program adapted to input and output information to a variety of local intelligent appliances at a high velocity and a high efficiency in the field of factory automation for efficiently controlling factory production lines or the field of process automation for controlling multiple industrial processes. Provided for this purpose is an improved a change-over system associated with a 1-bit processor and an ordinary processor for executing the sequence control program. Also improved are functions for a user to particularly specify a BASIC program, software/hardware control functions for a group of I/O boards for transferring signals to receiving signals from the local appliances, and the ability to program the internally stored sequence control program.

7 Claims, 29 Drawing Sheets

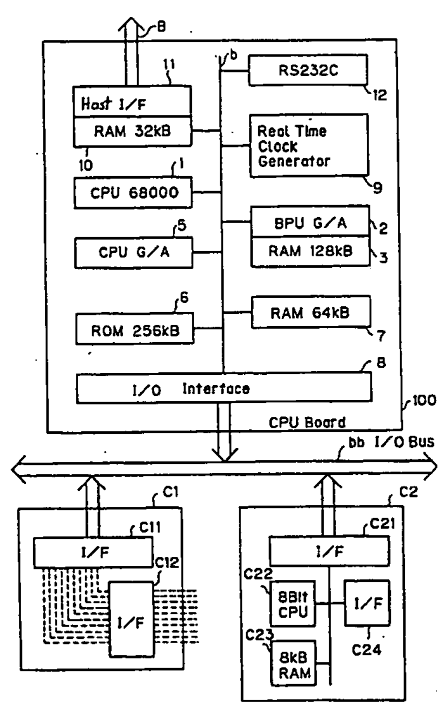
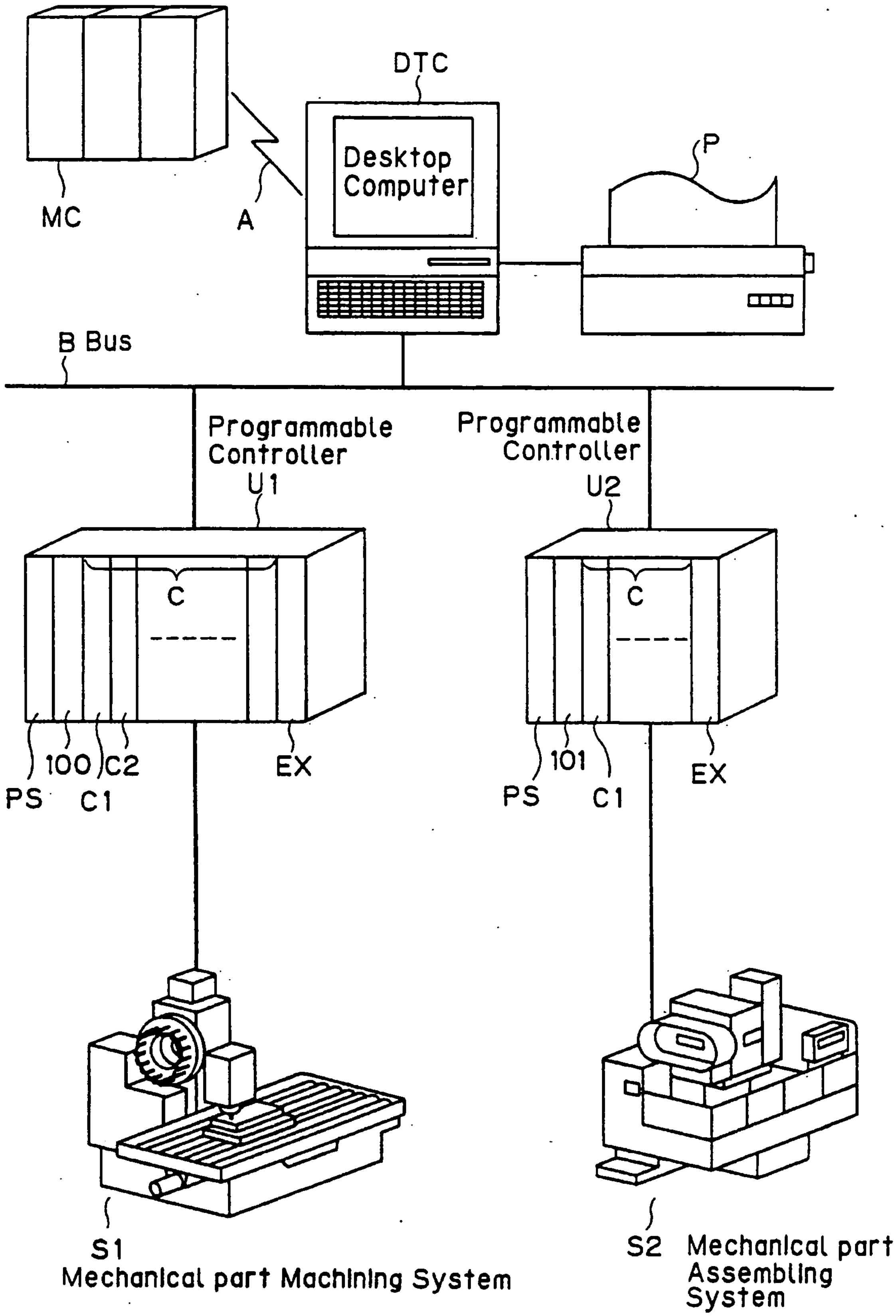


FIG.14



MC
A
DTC
Desktop Computer
P
B Bus
Programmable Controller
U1
Programmable Controller
U2
C
C
PS
100
C1
C2
EX
PS
101
C1
EX
S1
Mechanical part Machining System
S2
Mechanical part Assembling System

FIG. 15 is a diagram representing a software program in CPU board 100 that carries out parallel operations of sequence control processing SQ and BASIC processing BAS.

Sequence control processing SQ is intended to execute a string of program commands such as those shown in FIGS. 12 and 13, while BASIC processing BAS is associated with a data readout program which is particularly created by the user, or is a standard communication program, written in BASIC. An execution right change-over processing unit ES, a timer T, and a task scheduler TS have functions which are defined by software stored inside CPU board 100. Task scheduler TS functions to determine the priority for execution with respect to several processes in the BASIC processing program. This function is not, however, directly associated with the operations of the present invention.

It should be noted that generally several hundreds or several thousands of sequence control program commands are provided, and the execution time of one cycle of that program is several tens or hundreds of milliseconds (ms), depending largely on the configuration of the system to be controlled. For this example, regardless of what the execution time for one cycle is, it will be assumed that 10 ms is the time set in timer T. The time set in timer T is not limited to 10 ms but may be arbitrarily set depending on the system configuration.

As shown in the example of FIG. 15, the first step of sequence control processing SQ is to execute a load command LD. Subsequently, the execution moves to an AND command, an OR command, and an output command (OUT), at which time 10 ms passes. Then, a time-up signal is transmitted as an interrupt signal from timer T to execution right change-over processing unit ES. CPU 1 inputs this interrupt signal, but at this time the control executing right is still held by BPU 2. As a result, CPU 1 is unaffected by this time-up interrupt.

The sequence processing further advances, and BPU 2 executes load command LD. By the time the next 10 ms time-up interrupt signal is transmitted, the control executing right has been handed over to CPU 1 for execution of application command (1). Consequently, execution right change-over processing unit ES changes the processing which is to be executed by CPU 1 from sequence processing SQ to BASIC processing BAS. Thus, CPU 1 starts processing BASIC processing program BAS in accordance with task scheduler TS. Execution of sequence control processing SQ remains stopped.

Thereafter, timer T is again brought into a time-up state, and another 10 ms interrupt signal is generated. At this time, execution right change-over processing unit ES in CPU 1 temporarily stores the present conditions of the BASIC program (e.g., values of registers, value of a program counter, etc.) in data memory 7 (shown in FIG. 11), and causes the operation of sequence control processing SQ, which had been stopped, to resume.

When execution of sequence control processing SQ is completed, sequence control processing SQ transmits a notice of completion to execution right change-over processing unit ES, irrespective of whether a time-up signal has been issued, and CPU 1 initiates or resumes execution of BASIC processing BAS.

In sum, the system operates by alternating execution of sequence control processing SQ and BASIC processing BAS every 10 ms while CPU 1 has the execution right. As a result, sequence processing SQ and BASIC processing BAS appear to the user as if they were being executed simultaneously. Hence, multi-processing is practicable.

On the other hand, some system configurations do not need parallel execution of sequence control processing and BASIC processing. In some cases, either sequence control processing or BASIC processing is unnecessary, depending on modifications of the system. In that situation, CPU board 100 incorporates a program such as that shown in the flowchart of FIG. 16, and the system operates as follows.

After the power is turned ON and a self-diagnostic operation is executed, it is determined whether a working language is designated by the host computer. If not, as a default, the system proceeds as if the sequence language were specified: a system language table necessary for processing the sequence language is created in data memory 7 and sequence language processing begins. If a language is designated by the host computer, the program depicted in FIG. 16 checks to determine which language is specified. If the available language is a sequence language, the program proceeds as in the default situation described above. If the BASIC language is designated, a system table required for processing the BASIC language is created in data memory 7 and BASIC language processing begins. If both the sequence language and the BASIC language are designated, system tables necessary for processing both the sequence language and the BASIC language are created in data memory 7 and processing of both proceeds.

In this manner, the program creates the system tables corresponding to the languages designated, thereby facilitating the designation of the processing languages from the host computer, e.g., a desktop computer.

Next, a system as depicted in FIG. 17 will be described that uses a standard I/O driver set in CPU board 100 and thus is capable of utilizing various types of I/O boards.

An operating system OS in CPU board 100 receives an access request a1 for an I/O board C from a user program UP (such as a BASIC language program stored in CPU board 100) or an interrupt request a2 from I/O board C. Access request a1 from user program UP prompts an I/O request receiving process (1) to start, while interrupt request a2 prompts an interrupt request receiving process (2) to start.

When starting up the system, a standard I/O driver SD according to the present invention generates process definition tables TBL1, TBL2, . . . , TBLn for the individual I/O boards fitted to respective inter-unit slots. (Collectively, the tables are designated TBL; an arbitrary one of process definition tables TBL will be referred to as TBLi, i=1 to n.) When an I/O request receiving process (1) or an interrupt request receiving process (2) is begun, preparation for accessing the process definition tables TBL starts.

Upon initiating either process (1) or (2), a determination is made as to whether the I/O board to be accessed is classified as a register/interface type or a command/interface type, and then a corresponding main processing is executed. An example of a register/interface type I/O board is the ordinary I/O board C1 depicted in FIG. 11. The command/interface type I/O board is typified by I/O board C2 which, as illustrated in FIG. 11, incorporates a microprocessor to transfer and receive commands.

Common or standard operations for I/O request receiving process (1) and interrupt request receiving process (2) are performed as a common processing routine and are executed regardless of the type of interface of the I/O board. In addition, a special processing routine is executed for I/O boards requiring a different processing from the standard processing. At start-up, the special processing routine is uploaded to standard I/O driver SD from the I/O board that requires special processing. Also at start-up, an address of

FIG. 26 shows an example of the contents of comment file CF. FIG. 27 represents an example of the contents of circuit/comment corresponding table CCT. Comment file CF may also be stored in RAM 7 in CPU board 100 rather than in the programming tool. Additionally, a comment file which corresponds the step numbers to the subcomments may be added.

The operation of seeking out specific locations of the ladder circuits on the programming tool will be described with reference to FIG. 28. The programming tool reads the contents of comment file CF and circuit/comment corresponding table CCT, which is preset to indicate subsequent display pictures S1, S2, S3, and S4 on the CRT display unit.

An initial picture displayed on the CRT display unit of the programming tool is a circuit monitor menu picture S1, from which picture a circuit comment display picture S2 is selected. Then, a circuit comment list is displayed in a list format on the CRT. From the circuit comment display picture S2, a subcomment display picture S3 is further selected, and thereafter all the subcomments included in the circuit comment are displayed. When a circuit display picture S4 is selected corresponding to the subcomment, the ladder circuit corresponding to this subcomment is displayed.

In particular, in order to specify a certain ladder circuit from the ladder program shown in FIG. 24, a desired circuit comment is selected by displaying a list of the circuit comments, and a specific ladder circuit can be displayed on the CRT display by designating a ladder circuit corresponding to the subcomment included in this circuit comment.

Note that the circuit picture S4 may be selected from the circuit monitor menu picture S1 or the circuit comment list S2. Page update scrolling and page update monitoring can be effected on the respective display pictures.

As stated above, in a programmable controller according to the invention, the comment file and the circuit/comment corresponding table are set and then read out. A correspondence between the ladder circuits and a variety of comments added to the ladder circuits is established to enable a hierarchical display on a CRT. Thus, it is possible to immediately detect a desired ladder circuit.

The arrangement described above provides easy-to-search ladder circuits, useful for adjusting the circuits or locating an error. The description will next focus on improvements in creating the ladder program and on operation during debugging.

FIG. 29 shows a flowchart of a processing routine of a programmable controller according to the present invention. A sequence control process which uses a ladder program is executed by a processing routine consisting of common processing such as a self-diagnosis, I/O refresh processing of I/O registers in an I/O board, execution of the stored ladder program, and service processing for a host appliance.

In a programmable controller according to the invention, the I/O refresh process subsequent to the common process is, as illustrated in FIG. 29, omitted in order to perform programming and debugging when creating the sequence control program without mounting the I/O board. With this arrangement, the debugging operation can be done, without the I/O board, in conformity with an instruction from a debugger such as a programming tool or the like.

FIG. 30 is a diagram showing a programming function of the ladder program in a programmable controller according to the present invention. Respective blocks in the Figure represent software-functional blocks of the programmable controller.

The individual functional blocks in FIG. 30 work as follows. A circuit editing function 201 is for editing respective circuit components of the ladder circuit, with which a programmer describes addresses of the respective circuit components by using signal names similar to device names used when designing the ladder circuit. A signal defining function 202 serves as a unit for storing in a table format a correspondence of the addresses to the signal names of the respective circuit components. A compile function 203 transmits to a sequencer system 205 a program in an executable format with reference to the signal names in the ladder circuit. The corresponding addresses are supplied from signal defining function 202 and from an address automatic generating function 204. Address automatic generating function 204 automatically assigns detailed addresses to the signal names supplied from signal defining function 202.

Procedures for creating the ladder program by utilizing these functions are described below.

A ladder circuit such as that depicted in FIG. 31 is generated in cooperation with the programming tool and circuit editing function 201. The individual circuit components of a relay unit, an output unit, and so on are set in the form of signal names such as SW1 and COIL1. As shown in FIG. 32, signal defining function 202 arranges signal names SW1, SW2, COIL1, COIL2, IRL1, TIM1, CNT1 and REG1 to correspond to addresses X, X, Y, Y, I, T, C and D, where the symbol X is an address representing an input, Y is an output address, I is an internal relay address, T is a timer address, C is a counter address, and D is a data register address.

The detailed addresses are automatically assigned in address automatic generating function 204. For example, for the address X, set to circuit element SW1, a detailed address X001 is assigned. The detailed addresses are set sequentially to correspond to the number of the signal names of the ladder circuit components. In this example, the address X001 is set to the signal name SW1, and the address X002 is set to the signal name SW2. The results for all the components of FIG. 32 are shown in FIG. 33.

While processing is performed in sequencer system 205 in accordance with the program in executable format produced by compile function 203, debugging can be carried out solely by CPU board 100 without mounting the I/O board, because the I/O refresh process is, as illustrated in the flowchart of FIG. 29, bypassed if the I/O board is not mounted at the debugging stage. Based on the results of debugging, detailed addresses are added as needed.

Thus, the signal names can automatically correspond to the detailed addresses without a programmer being aware of the addresses of the respective ladder circuit components, which in turn facilitates a design of the sequence control program. Debugging can be accomplished without an I/O board, and the ladder program operations can be debugged before finishing the design of a relay board which corresponds to the sequence process.

A programmable controller according to the present invention typically performs programming with a ladder program having thousands of steps by splitting the number of blocks per step. FIGS. 34(*a*) through 34(*c*) show programming modes.

FIGS. 34(*a*), 34(*b*), and 34(*c*) represent steps 1, 2, and 3 to 5 of a part of a ladder program.

A start command "ACT PROG1.2" and an end command "INACT PROG1.1" are set at the final substep of step 1. With this arrangement, when a control operation of step 1

reaches the final substep, a block ladder program PROG1.2 of step 2 is started by stopping a block ladder program PROG1.1 of step 1, thereby initiating the control operation of step 2.

Set at the final substep of the block ladder program PROG1.2 are a stop command "INACT PROG1.2", for stopping the block ladder program PROG1.2, and parallel start commands "ACT PROG1.2", "ACT PROG2.1", and "ACT PROG3.1", for simultaneously starting steps 3 to 5.

The ends of steps 3 and 4 are monitored in step 5. When detecting the cessation of steps 3, 4, and 5, the operation returns to step 1, i.e., the starting step of the sequence control process, in response to stop command "INACT PROG3.1" and start command "ACT PROG1.1".

The start commands "ACT" and the stop commands "INACT" of the ladder program as thus defined make it possible to effect programming in parallel by splitting a series of thousands of sequence control programs into several blocks. In addition, the ladder programs inform each other of starting and ending, whereby the sequence control operation can be carried out by storing the ladder programs split into blocks in a plurality of CPU boards.

Turning to FIG. 35, an example is shown where a control object M (a product on a production line) on a control line L is controlled by combining a CPU board 101 in which a BASIC program is stored, and CPU boards 102 to 104 in which only ladder programs are stored.

Ladder programs LD1 and LD2 are stored in CPU board 102, to which an I/O board group C10 is connected; ladder programs LD3 through LD5 are stored in CPU board 103, to which an I/O board group C20 is connected; and a ladder program LD6 is stored in the CPU board 104, to which an I/O board group C30 is connected.

FIG. 36 illustrates an example of the BASIC program stored in CPU board 101.

Ladder programs LD1 through LD6 are defined as a series of sequence control programs with respect to control object M, these ladder programs having the block construction described above and being programmed independently.

According to the example shown in FIG. 36, when starting operation, the BASIC program stored in CPU board 101 sequentially actuates ladder programs LD1 and LD2 of CPU board 102. After these programs have ended, ladder program LD3 or LD4 of the CPU board 103 is executed in accordance with the sequence processing results at that time, indicated by COND1. Immediately after finishing the program LD3 or LD4, the BASIC program functions to start in parallel both ladder program LD5 incorporated into CPU board 103 and ladder program LD5 incorporated into CPU board 104.

In accordance with the illustrated system, sequence control programs in which a series of sequence control operations are divided into blocks are handled and processed by a plurality of programmable controllers. Thus, highly efficient sequence control processing can be done.

As discussed above, a programmable controller according to the invention is capable of improving the velocity of sequence control processing and attaining an easy-to-redesign system when modification is desired. A programmable controller exhibiting a high processing efficiency can thus be realized.

Although the illustrative embodiments of the present invention have been described in detail with reference to the accompanying drawings, it should be understood that the present invention is not limited to those precise embodiments. Various changes or modifications may be made by one skilled in the art without departing from the scope or spirit of the invention, and are limited only by the scope of the claims.

What is claimed is:

1. A programmable controller comprising:

one or more I/O boards for transferring and receiving multiple information to and from a control object; and

a processor board for imparting a control signal to the control object via the I/O boards, the processor board comprising:

first processor means for controlling the processor board, for executing commands of a sequence control program type, and for executing commands of a BASIC program type for performing general-purpose arithmetic processing, information processing, or a control operation, by starting executing the commands of sequence control program type and making an end instruction;

program memory means for storing the commands of sequence control program type;

1-bit processor means, connected directly to the program memory means, for executing commands to be executed that are sequentially read from the program memory means and transmitting the commands to be executed to the first processor means if the commands to be executed are of the sequence control program type to be executed by the first processor means;

a random access memory (RAM) for temporarily storing data;

a read only memory (ROM) for storing a self-diagnostic program;

a communication interface for communicating with a host computer;

an I/O bus through which I/O boards for transferring and receiving multiple information are connected;

an I/O interface connected to the I/O bus; and

an internal bus for mutually connecting the first processor means, the 1-bit processor means, the RAM, the ROM, the communication interface, and the I/O interface.

2. The programmable controller as claimed in claim 1, wherein at a start-up time an I/O driver stored in the processor board stores a process definition table for storing in a table format information read from an I/O board regarding a board ID, a type of interface of the I/O board, the number of channels, a command register address, a buffer address, a data register address, an address for designating special processing when special processing is needed, and wherein the I/O driver refers to the process definition table when effecting a data outputting process.

3. The programmable controller as claimed in claim 1, further comprising:

strobing signal generating means for generating strobing signals provided in the processor board and the I/O boards, wherein when starting a data transfer cycle, the processor board requests a data transfer by transmitting to one I/O board a number of strobing signals, each strobing signal making transmission of an information frame effective, and wherein the one I/O board receiving the data transfer request transmits to the processor board a number of other strobing signals, each strobing signal making transmission of an information frame effective, thus ending the data transfer cycle.

4. A programmable controller as claimed in claim 1, further comprising one or more abnormality detecting circuits that correspond to different kinds of abnormalities

which can arise, wherein the first processor means receives an abnormality detecting signal as an interrupt signal and stores, in a table format in memory, time data from an internal timer regarding the occurrence of the detected abnormality and a description of the detected abnormality.

5. A programmable controller as claimed in claim **1**, further comprising:

means for assigning signal names to circuit elements when creating ladder circuits;

means for corresponding the signal names to addresses in accordance with a preset signal name-to-address correspondence table; and

means for sequentially assigning detailed addresses corresponding to the signal names,

wherein an I/O refresh process in a sequence control process routine is omitted when an I/O board is not connected with the controller.

6. A programmable controller as claimed in claim **1**, wherein a series of sequence control operations are split into blocks corresponding to several steps when programming a ladder program, and wherein the ladder program is stored and executed block by block by storing in a final substep of each block a command for stopping the execution of that block and a command for specifying the block to be executed next.

7. A programmable controller as claimed in claim **1**, further comprising:

means for storing a comment file for storing comments added to ladder circuits in a ladder program generated in a ladder language and corresponding step numbers of the ladder circuit concerned; and

means for storing a circuit/comment table in which the step numbers of the ladder circuit concerned correspond to comment numbers in the comment file,

wherein the circuit comments, the step numbers and the ladder circuits are read from a programming tool.

* * * * *

Subscribe (Full Service) Register (Limited Service, Free) Login

Search: ◉ The ACM Digital Library ○ The Guide

+optimizing, +application program, +routine,processes

USPTO

Feedback Report a problem Satisfaction survey

Terms used **optimizing application program routine processes**

Found **27** of **166,357**

Sort results by: relevance

Display results: expanded form

Save results to a Binder

Search Tips

☐ Open results in a new window

Try an Advanced Search

Try this search in The ACM Guide

Results 1 - 20 of 27

Result page: **1** 2 next

Relevance scale

1 Compilation: Efficient static single assignment form for predication

Arthur Stoutchinin, Francois de Ferriere

December 2001 **Proceedings of the 34th annual ACM/IEEE international symposium on Microarchitecture**

Publisher: IEEE Computer Society

Full text available: pdf(961.27 KB) Publisher Site

Additional Information: full citation, abstract, references

We present a framework that allows translation of predicated code into the static single assignment (SSA) form, and simplifies application of the SSA-based optimizations to predicated code. In particular, we represent predicate join points in the program by the Ψ-functions similar to the Φ-functions of the basic SSA. The SSA-based optimizations (such as constant propagation) can be applied to predicated code by simply specifying additional rules for processing the Ψ-functions. We pre ...

2 Real-time tasking semantics working group

John Goodenough

May 1989 **ACM SIGAda Ada Letters , Proceedings of the third international workshop on Real-time Ada issues IRTAW '89**, Volume X Issue 4

Publisher: ACM Press

Full text available: pdf(1.36 MB)

Additional Information: full citation, citings, index terms

3 Automatic discovery of parallelism: a tool and an experiment (extended abstract)

Michael Burke, Ron Cytron, Jeanne Ferrante, Wilson Hsieh, Vivek Sarkar, David Shields

January 1988 **ACM SIGPLAN Notices , Proceedings of the ACM/SIGPLAN conference on Parallel programming: experience with applications, languages and systems PPEALS '88**, Volume 23 Issue 9

Publisher: ACM Press

Full text available: pdf(887.45 KB)

Additional Information: full citation, abstract, references, citings, index terms

This paper reports preliminary results from applying advanced techniques to the parallelization of sequential programs. Such techniques include interprocedural analysis and the identification of nested parallelism. These techniques have been proposed for exploiting the greater concurrency offered by multiprocessors as compared with vector architectures. The effectiveness of these techniques as applied to some popular numerical Fortran programs is examined. Although extrapolation is difficul ...

4 Efficient simulation of caches under optimal replacement with applications to miss characterization
Rabin A. Sugumar, Santosh G. Abraham
June 1993 **ACM SIGMETRICS Performance Evaluation Review , Proceedings of the 1993 ACM SIGMETRICS conference on Measurement and modeling of computer systems SIGMETRICS '93**, Volume 21 Issue 1
Publisher: ACM Press
Full text available: pdf(1.26 MB) Additional Information: full citation, abstract, references, citings, index terms

Cache miss characterization models such as the three Cs model are useful in developing schemes to reduce cache misses and their penalty. In this paper we propose the OPT model that uses cache simulation under optimal (OPT) replacement to obtain a finer and more accurate characterization of misses than the three Cs model. However, current methods for optimal cache simulation are slow and difficult to use. We present three new techniques for optimal cache simulation. First, we propose a limited lo ...

5 Using Ada for PC-based software development
Charles R. Snyder
June 1991 **Proceedings of the eighth annual Washington Ada symposium & summer SIGAda meeting on Ada: software: foundation for competitveness**
Publisher: ACM Press
Full text available: pdf(892.46 KB) Additional Information: full citation, references, index terms

6 Slice-processors: an implementation of operation-based prediction
Andreas Moshovos, Dionisios N. Pnevmatikatos, Amirali Baniasadi
June 2001 **Proceedings of the 15th international conference on Supercomputing**
Publisher: ACM Press
Full text available: pdf(236.51 KB) Additional Information: full citation, abstract, references, citings, index terms

We describe the Slice Processor micro-architecture that implements a generalized operation-based prefetching mechanism. Operation-based prefetchers predict the series of operations, or the computation slice that can be used to calculate forthcoming memory references. This is in contrast to outcome-based predictors that exploit regularities in the (address) outcome stream. Slice processors are a generalization of existing operation-based prefetching mechanisms such as stream buffers where the ...

7 Computer System Simulation: An Introduction
M. H. MacDougall
September 1970 **ACM Computing Surveys (CSUR)**, Volume 2 Issue 3
Publisher: ACM Press
Full text available: pdf(1.26 MB) Additional Information: full citation, references, citings, index terms

8 Interprocedural dependence analysis and parallelization
Michael Burke, Ron Cytron
July 1986 **ACM SIGPLAN Notices , Proceedings of the 1986 SIGPLAN symposium on Compiler contruction SIGPLAN '86**, Volume 21 Issue 7
Publisher: ACM Press
Full text available: pdf(1.74 MB) Additional Information: full citation, abstract, references, citings, index terms

We present a method that combines a deep analysis of program dependences with a broad analysis of the interaction among procedures. The method is more efficient than existing methods: we reduce many tests, performed separately by existing methods, to a single test. The method is more precise than existing methods with respect to references to multi-dimensional arrays and dependence information hidden by procedure calls. The method is more general than existing methods: we accommodate potent ...

9 Metadatabase solutions for enterprise information integration problems
Cheng Hsu, Laurie Rattner
January 1993 **ACM SIGMIS Database**, Volume 24 Issue 1
Publisher: ACM Press
Full text available: pdf(1.29 MB) Additional Information: full citation, abstract, index terms

The success of modern information technology in the past decades has brought about the proliferation of systems dedicated to individual groups of applications and functions. This proliferation, in turn, has led to the need for enterprise-wide management and integration of information, and has triggered major efforts such as systems integration, re-engineering, and computer integrated manufacturing. Nonetheless, achieving such integration remains a challenge.To effectively manage information reso ...

10 An Elementary Discussion of Compiler/Interpreter Writing
R. L. Glass
January 1969 **ACM Computing Surveys (CSUR)**, Volume 1 Issue 1
Publisher: ACM Press
Full text available: pdf(1.85 MB) Additional Information: full citation, references, citings, index terms

11 The impact of mesa on system design
Hugh C. Lauer, Edwin H. Satterthwaite
September 1979 **Proceedings of the 4th international conference on Software engineering**
Publisher: IEEE Press
Full text available: pdf(947.74 KB) Additional Information: full citation, abstract, references, citings, index terms

The Mesa programming language supports program modularity in ways that permit subsystems to be developed separately but to be bound together with complete type safety. Separate and explicit interface definitions provide an effective means of communication, both between programs and between programmers. A configuration language describes the organization of a system and controls the scopes of interfaces. These facilities have had a profound impact on the way we design systems and organize de ...

12 Parallel expert system search techniques for a real-time application
G. B. Lamont, D. J. Shakley
January 1989 **Proceedings of the third conference on Hypercube concurrent computers and applications - Volume 2**
Publisher: ACM Press
Full text available: pdf(1.14 MB) Additional Information: full citation, abstract, references, citings, index terms

Expert systems are being used to govern the intelligent control of the Robotic Air Vehicle (RAV) which is currently a research project at the Air Force Avionics Laboratory. Due to the nature of the RAV system the associated expert system needs to perform in a demanding real-time environment. The use of a parallel processing capability to support the associated computational requirement may be critical in this application. Thus, parallel

search algorithms for real-time expert systems are des ...

13 [The cost of conservative synchronization in parallel discrete event simulations]
David M. Nicol
April 1993 **Journal of the ACM (JACM)**, Volume 40 Issue 2
Publisher: ACM Press
Full text available: pdf(2.11 MB) Additional Information: full citation, abstract, references, citings, index terms, review

This paper analytically studies the performance of a synchronous conservative parallel discrete-event simulation protocol. The class of models considered simulates activity in a physical domain, and possesses a limited ability to predict future behavior. Using a stochastic model, it is shown that as the volume of simulation activity in the model increases relative to a fixed architecture, the complexity of the average per-event overhead due to synchronization, event list manipulation, looka ...

Keywords: conservative synchronization

14 Hierarchical storage in information retrieval
John Salasin
May 1973 **Communications of the ACM**, Volume 16 Issue 5
Publisher: ACM Press
Full text available: pdf(460.87 KB) Additional Information: full citation, abstract, references, citings

A probabilistic analysis is employed to determine the effect of hierarchical storage organizations on information retrieval operations. The data storage hardware is assumed to consist of n-levels of linearly connected memory hardware with increasing data access times and increasing data storage capabilities. A system might, for example, consist of fast semiconductor memory, computer core memory, extended core storage, disk memory, and data cells. Equations are derived to pr ...

Keywords: hierarchical storage, information retrieval

15 Robustness: CMC: a pragmatic approach to model checking real code
Madanlal Musuvathi, David Y. W. Park, Andy Chou, Dawson R. Engler, David L. Dill
December 2002 **ACM SIGOPS Operating Systems Review**, Volume 36 Issue SI
Publisher: ACM Press
Full text available: pdf(1.55 MB) Additional Information: full citation, abstract, references

Many system errors do not emerge unless some intricate sequence of events occurs. In practice, this means that most systems have errors that only trigger after days or weeks of execution. Model checking [4] is an effective way to find such subtle errors. It takes a simplified description of the code and exhaustively tests it on all inputs, using techniques to explore vast state spaces efficiently. Unfortunately, while model checking systems code would be wonderful, it is almost never done in pra ...

16 Cloud pattern recognition
R. D. Joseph, S. S. Viglione, H. F. Wolf
January 1964 **Proceedings of the 1964 19th ACM national conference**
Publisher: ACM Press
Full text available: pdf(1.47 MB) Additional Information: full citation, abstract, references, index terms

Cloud cover photographs transmitted from meteorological satellites must be processed and interpreted before weather maps can be issued. Most of the routine processing can be

handled by present day digital computer techniques; however, the recognition and interpretation of cloud patterns such as vortices indicating hurricanes, must still be performed by humans due to the lack of suitable recognition mechanisms. This paper investigates the feasibility of using a perceptron-type computer for t ...

17 Toward real time simulation: prototyping of a large scale parallel ground target simulation
John B. Gilmer, David W. O'Brien, Jeffery E. Payne
December 1990 **Proceedings of the 22nd conference on Winter simulation**
Publisher: IEEE Press
Full text available: pdf(878.73 KB) Additional Information: full citation, references, citings, index terms

18 A general purpose design automation file system
T. Beretvas, C. H. Liu, R. L. Taylor
January 1967 **Proceedings of the 4th conference on Design automation**
Publisher: ACM Press
Full text available: pdf(593.50 KB) Additional Information: full citation, abstract, citings, index terms

This file system grew out of exploratory work in the file system-terminal area. The main reason for such a project was to explore file organizations that would be useful for a wide range of Design Automation applications. The specific problem we chose was the update and maintenance of logic sheets. A secondary reason was to gain some knowledge of how the standard capabilities of OS/360 could be used to implement a file system.

19 Characterization of alpha AXP performance using TP and SPEC workloads
Z. Cvetanovic, D. Bhandarkar
April 1994 **ACM SIGARCH Computer Architecture News , Proceedings of the 21ST annual international symposium on Computer architecture ISCA '94**, Volume 22 Issue 2
Publisher: IEEE Computer Society Press, ACM Press
Full text available: pdf(1.00 MB) Additional Information: full citation, abstract, references, citings, index terms

The characteristics of several commercial and technical workloads on the DEC 7000 AXP system are compared using built-in hardware monitors. The data analyzed include total instructions, cycles, multiple-issued instructions, stall components, cache misses, and instruction types. The data indicates that the two classes of workloads have vastly different characteristics and impose different requirements on the system design. Compared to VAX, Alpha AXP takes advantage of lower cycles per instruction ...

20 The preconditioned conjugate gradient method on the hypercube
G. Abe, K. Hane
January 1989 **Proceedings of the third conference on Hypercube concurrent computers and applications - Volume 2**
Publisher: ACM Press
Full text available: pdf(814.33 KB) Additional Information: full citation, abstract, references, citings, index terms

A parallel algorithm for solving the elliptic partial differential equation (PDE) is described in this paper through the finite difference method (FDM) The Concurrent Preconditioned Conjugate Gradient method is developed to optimize processor load balancing. This algorithm is evaluated on a hypercube-based concurrent machine, the Intel iPSC.

Ref #	Hits	Search Query	DBs	Default Operator	Plurals	Time Stamp
S1	3	(obtain$3 or acquir$3 or ge$2 or gain$3) same (sort$3 or filter$3 or sift$3) same (performance near2 data) same too$2 and (@ad<"20010723" or @prad<"20010723" or @rlad<"20010723")	US-PGPUB; USPAT; USOCR; EPO; JPO; DERWENT; IBM_TDB	OR	OFF	2005/06/23 13:39
S2	93	(obtain$3 or acquir$3 or ge$2 or gain$3) same (sort$3 or filter$3 or sift$3) same (performance near2 data) and too$2 and (@ad<"20010723" or @prad<"20010723" or @rlad<"20010723")	US-PGPUB; USPAT; USOCR; EPO; JPO; DERWENT; IBM_TDB	OR	OFF	2005/06/23 13:39
S3	3	(obtain$3 or acquir$3 or ge$2 or gain$3) same (sort$3 or filter$3 or sift$3) same (performance near2 data) same (priorit$5 or arrang$3) and too$2 and (@ad<"20010723" or @prad<"20010723" or @rlad<"20010723")	US-PGPUB; USPAT; USOCR; EPO; JPO; DERWENT; IBM_TDB	OR	OFF	2005/06/23 13:41
S4	18	(obtain$3 or acquir$3 or ge$2 or gain$3) same (sort$3 or filter$3 or sift$3) same (performance near2 data) same(priorit$5 or priority or group$3 or arrang$3) and too$2 and (@ad<"20010723" or @prad<"20010723" or @rlad<"20010723")	US-PGPUB; USPAT; USOCR; EPO; JPO; DERWENT; IBM_TDB	OR	OFF	2005/06/23 13:43
S5	0	(obtain$3 or acquir$3 or ge$2 or gain$3) same (sort$3 or filter$3 or sift$3) same (performance near2 data) same(priorit$5 or priority or group$3 or arrang$3) and too$2 and (GUI or UI) and (@ad<"20010723" or @prad<"20010723" or @rlad<"20010723")	US-PGPUB; USPAT; USOCR; EPO; JPO; DERWENT; IBM_TDB	OR	OFF	2005/06/23 13:43
S6	1	(obtain$3 or acquir$3 or ge$2 or gain$3) same (sort$3 or filter$3 or sift$3) same (performance near2 data) same(priorit$5 or priority or group$3 or arrang$3) and too$2 and (GUI or UI or ((graphical near2 user) near2 interfac$2)) and (@ad<"20010723" or @prad<"20010723" or @rlad<"20010723")	US-PGPUB; USPAT; USOCR; EPO; JPO; DERWENT; IBM_TDB	OR	OFF	2005/06/23 13:47

S7	13	(obtain$3 or acquir$3 or ge$2 or gain$3) same (sort$3 or filter$3 or sift$3) same (performance near2 data) and (priorit$5 or priority or group$3 or arrang$3) and too$2 and (GUI or UI or ((graphical near2 user) near2 interfac$2)) and (@ad<"20010723" or @prad<"20010723" or @rlad<"20010723")	US-PGPUB; USPAT; USOCR; EPO; JPO; DERWENT; IBM_TDB	OR	OFF	2005/06/23 13:59
S8	0	(obtain$3 or acquir$3 or ge$2 or gain$3) same (sort$3 or filter$3 or sift$3) same (performance near2 data) same (program or modul$2 or fil$2) and (priorit$5 or priority or group$3 or arrang$3) and too$2 and (GUI or UI or ((graphical near2 user) near2 interfac$2)) and (@ad<"20010723" or @prad<"20010723" or @rlad<"20010723")	US-PGPUB; USPAT; USOCR; EPO; JPO; DERWENT; IBM_TDB	OR	OFF	2005/06/23 14:00
S9	13	(obtain$3 or acquir$3 or ge$2 or gain$3) same (sort$3 or filter$3 or sift$3) same (performance near2 data) and (priorit$5 or priority or group$3 or arrang$3) and too$2 and (GUI or UI or ((graphical near2 user) near2 interfac$2))and (program or modul$2 or fil$2) and (@ad<"20010723" or @prad<"20010723" or @rlad<"20010723")	US-PGPUB; USPAT; USOCR; EPO; JPO; DERWENT; IBM_TDB	OR	OFF	2005/06/23 14:10
S10	0	(obtain$3 or acquir$3 or ge$2 or gain$3) same (sort$3 or filter$3) same (performance near2 data) same (program or fil$2) and too$2 and (GUI or UI or ((graphical near2 user) near2 interfac$2))and (program or modul$2 or fil$2) and (@ad<"20010723" or @prad<"20010723" or @rlad<"20010723")	US-PGPUB; USPAT; USOCR; EPO; JPO; DERWENT; IBM_TDB	OR	OFF	2005/06/23 14:12
S11	1428	(obtain$3 or acquir$3 or ge$2 or gain$3) same (sort$3 or filter$3) same (data) and too$2 and (GUI or UI or ((graphical near2 user) near2 interfac$2))and (program or modul$2 or fil$2) and (@ad<"20010723" or @prad<"20010723" or @rlad<"20010723")	US-PGPUB; USPAT; USOCR; EPO; JPO; DERWENT; IBM_TDB	OR	OFF	2005/06/23 14:14

S12	50	(obtain$3 or acquir$3 or ge$2 or gain$3) same (sort$3 or filter$3) same (data) same (contex$2) and too$2 and (GUI or UI or ((graphical near2 user) near2 interfac$2))and (program or modul$2 or fil$2) and processo$2 and (@ad<"20010723" or @prad<"20010723" or @rlad<"20010723")	US-PGPUB; USPAT; USOCR; EPO; JPO; DERWENT; IBM_TDB	OR	OFF	2005/06/23 14:18
S13	12	(obtain$3 or acquir$3 or ge$2 or gain$3) same (sort$3 or filter$3) same (data) same (contex$2) and too$2 and (GUI or UI or ((graphical near2 user) near2 interfac$2))and (program or modul$2 or fil$2) and (performance near2 data) and processo$2 and (@ad<"20010723" or @prad<"20010723" or @rlad<"20010723")	US-PGPUB; USPAT; USOCR; EPO; JPO; DERWENT; IBM_TDB	OR	OFF	2005/06/23 14:24
S14	7	(obtain$3 or acquir$3 or ge$2 or gain$3) same (performance near2 data) same (sort$3 or filter$3) and too$2 and (GUI or UI or ((graphical near2 user) near2 interfac$2)) and (program or modul$2 or fil$2) and processo$2 and (@ad<"20010723" or @prad<"20010723" or @rlad<"20010723")	US-PGPUB; USPAT; USOCR; EPO; JPO; DERWENT; IBM_TDB	OR	OFF	2005/06/23 14:29
S15	2	(obtain$3 or acquir$3 or ge$2 or gain$3) same (performance near2 data) same (syste$2) same (sort$3 or filter$3) and too$2 and (GUI or UI or ((graphical near2 user) near2 interfac$2)) and processo$2 and (@ad<"20010723" or @prad<"20010723" or @rlad<"20010723")	US-PGPUB; USPAT; USOCR; EPO; JPO; DERWENT; IBM_TDB	OR	OFF	2005/06/23 14:43
S16	81	(obtain$3 or acquir$3 or gain$3) same (performance near2 data) same (system$2) and (sort$3 or filter$3) and too$2 and (GUI or UI or ((graphical near2 user) near2 interfac$2)) and processo$2 and (@ad<"20010723" or @prad<"20010723" or @rlad<"20010723")	US-PGPUB; USPAT; USOCR; EPO; JPO; DERWENT; IBM_TDB	OR	OFF	2005/11/02 09:24

S17	5	(obtain$3 or acquir$3 or gain$3) same (performance near2 data) same (system$2) and (sort$3 or filter$3) and too$2 and (GUI or UI or ((graphical near2 user) near2 interfac$2)) and "717"/$.ccls. and processo$2 and (@ad<"20010723" or @prad<"20010723" or @rlad<"20010723")	US-PGPUB; USPAT; USOCR; EPO; JPO; DERWENT; IBM_TDB	OR	OFF	2005/06/24 11:44
S18	0	(obtain$3 or acquir$3 or gain$3) same (performance near2 data) same (system$2) and (sort$3 or filter$3) and too$2 and (GUI or UI or ((graphical near2 user) near2 interfac$2)) and "717"/$.ccls. and advice and processo$2 and (@ad<"20010723" or @prad<"20010723" or @rlad<"20010723")	US-PGPUB; USPAT; USOCR; EPO; JPO; DERWENT; IBM_TDB	OR	OFF	2005/06/24 11:45
S19	0	(obtain$3 or acquir$3 or gain$3) same (performance near2 data) same (system$2) and (sort$3 or filter$3) and too$2 and (GUI or UI or ((graphical near2 user) near2 interfac$2)) and "717"/$.ccls. and advic$3 and processo$2 and (@ad<"20010723" or @prad<"20010723" or @rlad<"20010723")	US-PGPUB; USPAT; USOCR; EPO; JPO; DERWENT; IBM_TDB	OR	OFF	2005/06/24 11:45
S20	3	(obtain$3 or acquir$3 or gain$3) same (performance near2 data) same (system$2) and (sort$3 or filter$3) and too$2 and (GUI or UI or ((graphical near2 user) near2 interfac$2)) and "717"/$.ccls. and insight and processo$2 and (@ad<"20010723" or @prad<"20010723" or @rlad<"20010723")	US-PGPUB; USPAT; USOCR; EPO; JPO; DERWENT; IBM_TDB	OR	OFF	2005/06/24 11:45
S21	15	(obtain$3 or acquir$3 or gain$3) same (performance near2 data) same (system$2) and (sort$3 or filter$3) and (too$2 same insight) and (GUI or UI or ((graphical near2 user) near2 interfac$2)) and processo$2 and (@ad<"20010723" or @prad<"20010723" or @rlad<"20010723")	US-PGPUB; USPAT; USOCR; EPO; JPO; DERWENT; IBM_TDB	OR	OFF	2005/11/02 09:27

S22	20	(obtain$3 or acquir$3 or gain$3) same (performance near2 data) same (system$2) and (sort$3 or filter$3) and too$2 and insight and (GUI or UI or ((graphical near2 user) near2 interfac$2)) and processo$2 and (@ad<"20010723" or @prad<"20010723" or @rlad<"20010723")	US-PGPUB; USPAT; USOCR; EPO; JPO; DERWENT; IBM_TDB	OR	OFF	2005/11/02 09:28
S23	0	(obtain$3 or acquir$3 or gain$3) same (performance near2 data) same (system$2) and (sort$3 or filter$3) and too$2 and insight and (GUI or UI or ((graphical near2 user) near2 interfac$2)) and clocktic$2 and processo$2 and (@ad<"20010723" or @prad<"20010723" or @rlad<"20010723")	US-PGPUB; USPAT; USOCR; EPO; JPO; DERWENT; IBM_TDB	OR	OFF	2005/11/02 09:28